EXHIBIT C

US 20190095173A1

(19) United States

(12) Patent Application Publication

(10) Pub. No.: US 2019/0095173 A1

(43) Pub. Date: Mar. 28, 2019

Kaufman et al.

(54) SYSTEMS AND METHODS FOR AUTOMATICALLY GENERATING USER INTERFACE ELEMENTS FOR COMPLEX DATABASES

(71) Applicant: **Michael Philip Kaufman**, New York, NY (US)

(72) Inventors: **Michael Philip Kaufman**, New York, NY (US); **Micah Philip Silverman**, Huntington Station, NY (US)

(21) Appl. No.: **16/034,696**

(22) Filed: **Jul. 13, 2018**

Related U.S. Application Data

(60) Continuation of application No. 14/324,414, filed on Jul. 7, 2014, now Pat. No. 10,025,801, which is a continuation of application No. 13/385,913, filed on Mar. 14, 2012, now Pat. No. 8,775,478, which is a continuation of application No. 12/930,849, filed on Jan. 19, 2011, now Pat. No. 8,161,081, which is a division of application No. 11/925,236, filed on Oct. 26, 2007, now Pat. No. 7,885,981, which is a continuation of application No. 10/428,209, filed on Apr. 30, 2003, now Pat. No. 7,318,066, which is a continuation of application No. PCT/US2001/042867, filed on Oct. 31, 2001.

(60) Provisional application No. 60/276,385, filed on Mar. 16, 2001.

Publication Classification

(51) **Int. Cl.**

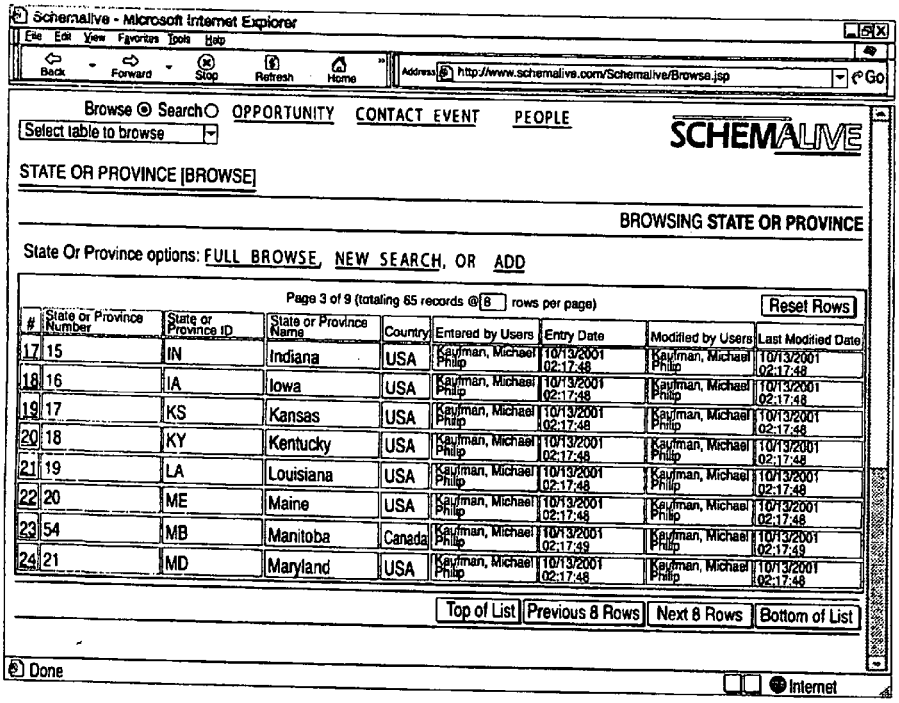
G06F 7/00	(2006.01)
G06F 3/0484	(2013.01)
G06F 3/0481	(2013.01)

(52) **U.S. Cl.**

CPC ***G06F 7/00*** (2013.01); *Y10S 707/912* (2013.01); ***G06F 3/0481*** (2013.01); ***G06F 3/0484*** (2013.01)

(57) **ABSTRACT**

In one embodiment, a software system automatically generates a fully functional user interface (UI) based upon any underlying schema within a relational database management system (RDBMS). The UI derives from an automated interrogation of the schema, and comprises all mode displays (e.g., browse, search, edit, add) for all tables, along with integrated mechanisms for representing, navigating and managing relationships across tables. It utilizes a hierarchical "context stack" for suspending the working state of a particular table while "drilling down" to work with related-table information and (potentially) return relevant changes to the base table. The UI presentation resolves cross-table relationships so as to supplant internal key fields from the primary table with corresponding descriptive fields derived from the related tables. Techniques are also provided to enhance and extend the internal representation of table structures, constraints, relationships and special requirements ("business rules") for improved discovery of the schema structure via automated interrogation.



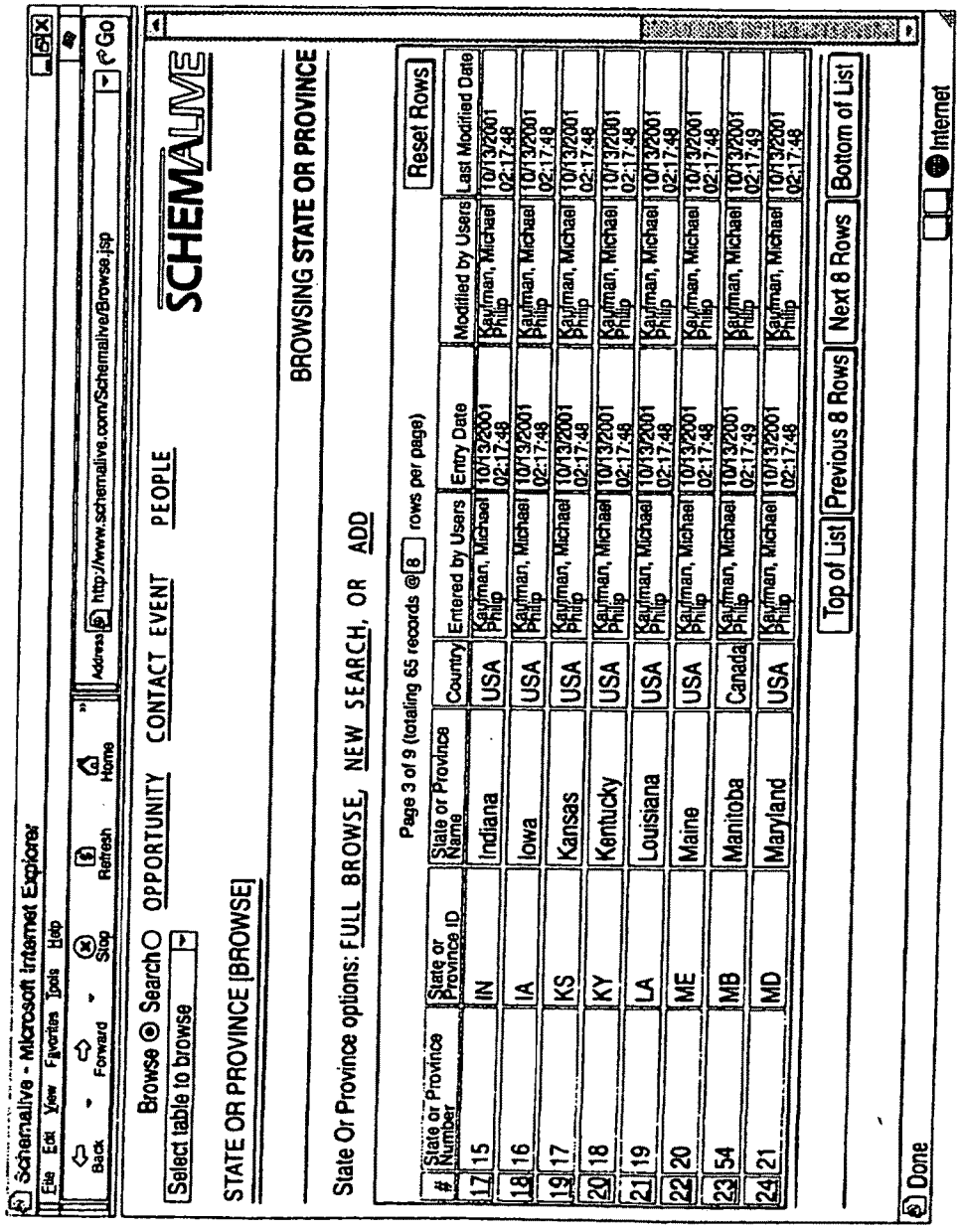

Schemalive - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://www.schemalive.com/Schemalive/Browse.jsp
Go
Browse Search OPPORTUNITY CONTACT EVENT PEOPLE
SCHEMALIVE
Select table to browse
STATE OR PROVINCE [BROWSE]
BROWSING STATE OR PROVINCE
State Or Province options: FULL BROWSE, NEW SEARCH, OR ADD
Page 3 of 9 (totaling 65 records @ 8 rows per page)
Reset Rows
| State or Province Number | State or Province ID | State or Province Name | Country | Entered by Users | Entry Date | Modified by Users | Last Modified Date
17 | 15 | IN | Indiana | USA | Kaufman, Michael Philip | 10/13/2001 02:17:48 | Kaufman, Michael Philip | 10/13/2001 02:17:48
18 | 16 | IA | Iowa | USA | Kaufman, Michael Philip | 10/13/2001 02:17:48 | Kaufman, Michael Philip | 10/13/2001 02:17:48
19 | 17 | KS | Kansas | USA | Kaufman, Michael Philip | 10/13/2001 02:17:48 | Kaufman, Michael Philip | 10/13/2001 02:17:48
20 | 18 | KY | Kentucky | USA | Kaufman, Michael Philip | 10/13/2001 02:17:48 | Kaufman, Michael Philip | 10/13/2001 02:17:48
21 | 19 | LA | Louisiana | USA | Kaufman, Michael Philip | 10/13/2001 02:17:48 | Kaufman, Michael Philip | 10/13/2001 02:17:48
22 | 20 | ME | Maine | USA | Kaufman, Michael Philip | 10/13/2001 02:17:48 | Kaufman, Michael Philip | 10/13/2001 02:17:48
23 | 54 | MB | Manitoba | Canada | Kaufman, Michael Philip | 10/13/2001 02:17:49 | Kaufman, Michael Philip | 10/13/2001 02:17:49
24 | 21 | MD | Maryland | USA | Kaufman, Michael Philip | 10/13/2001 02:17:48 | Kaufman, Michael Philip | 10/13/2001 02:17:48
Top of List
Previous 8 Rows
Next 8 Rows
Bottom of List
Done
Internet

FIG. 1

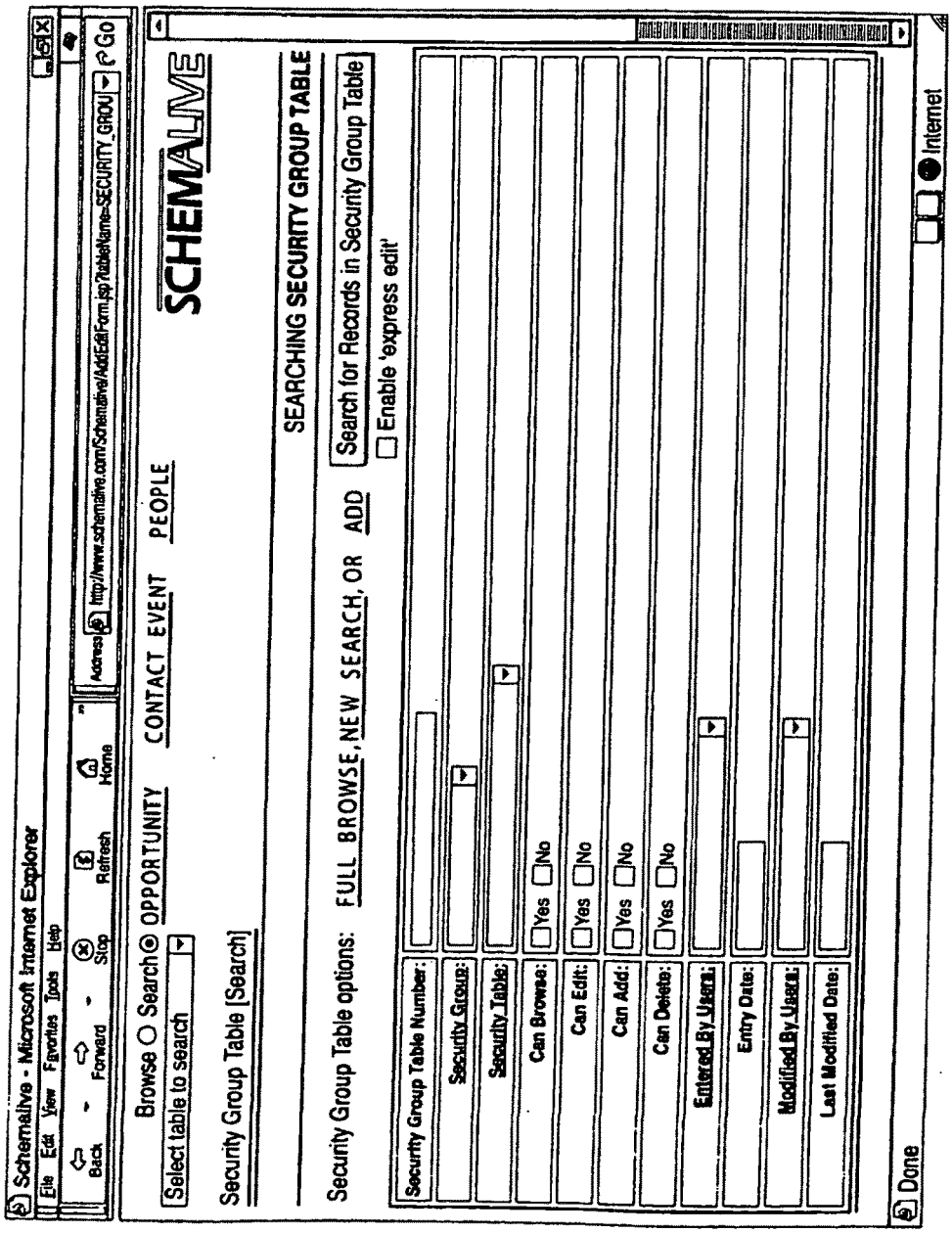

Schemalive - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://www.schemalive.com/Schemalive/AddEditForm.jsp?tableName=SECURITY_GROU
Go
Browse Search OPPORTUNITY CONTACT EVENT PEOPLE
SCHEMALIVE
Select table to search
Security Group Table [Search]
SEARCHING SECURITY GROUP TABLE
Security Group Table options: FULL BROWSE, NEW SEARCH, OR ADD
Search for Records in Security Group Table
Enable 'express edit'
Security Group Table Number:
Security Group:
Security Table:
Can Browse: Yes No
Can Edit: Yes No
Can Add: Yes No
Can Delete: Yes No
Entered By Users:
Entry Date:
Modified By Users:
Last Modified Date:
Done
Internet

FIG. 2

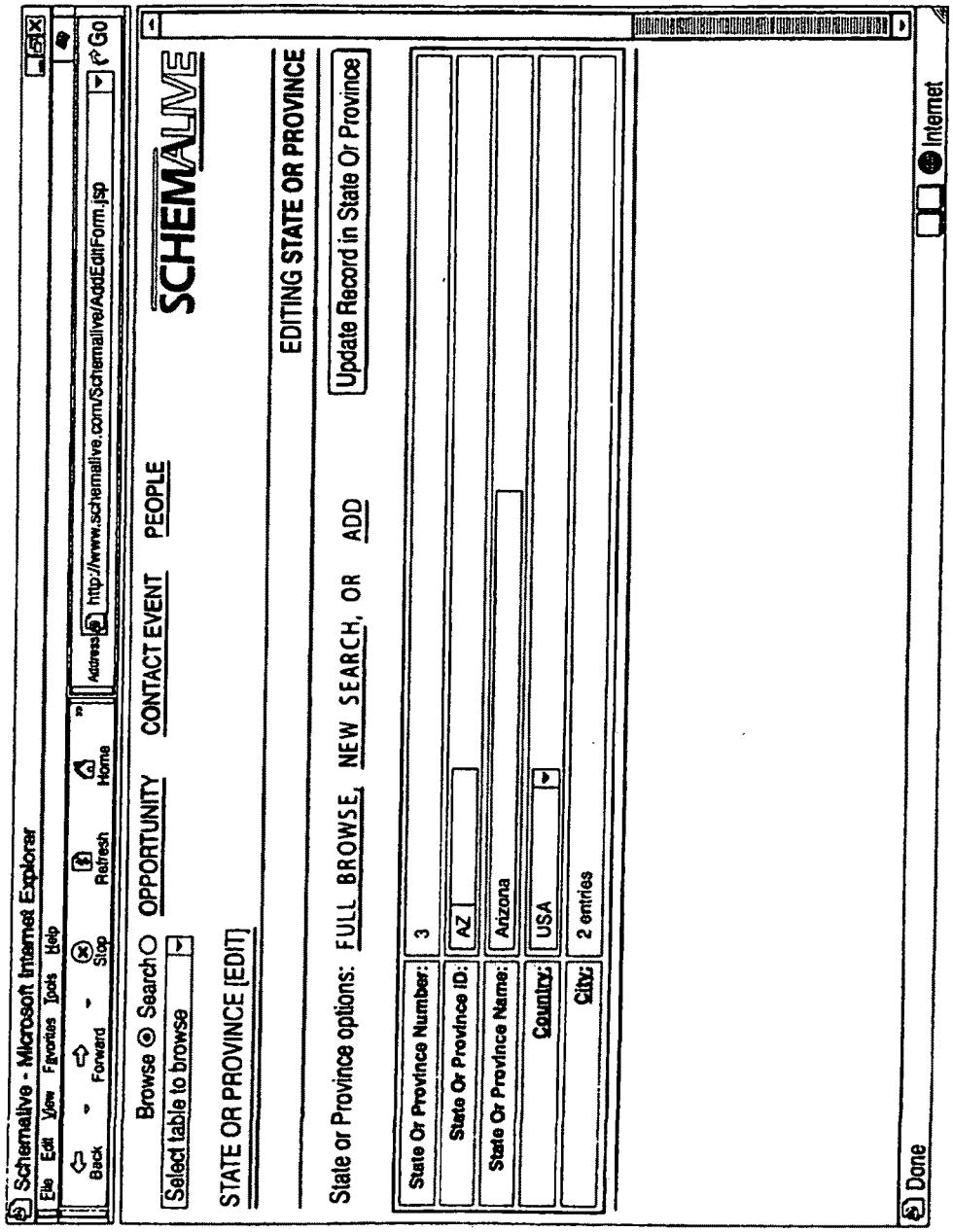

Schemalive - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://www.schemalive.com/Schemalive/AddEditForm.jsp
Go
Browse Search OPPORTUNITY CONTACT EVENT PEOPLE
SCHEMALIVE
Select table to browse
STATE OR PROVINCE [EDIT]
EDITING STATE OR PROVINCE
State or Province options: FULL BROWSE, NEW SEARCH, OR ADD
Update Record in State Or Province
State Or Province Number: 3
State Or Province ID: AZ
State Or Province Name: Arizona
Country: USA
City: 2 entries
Done
Internet

FIG. 3

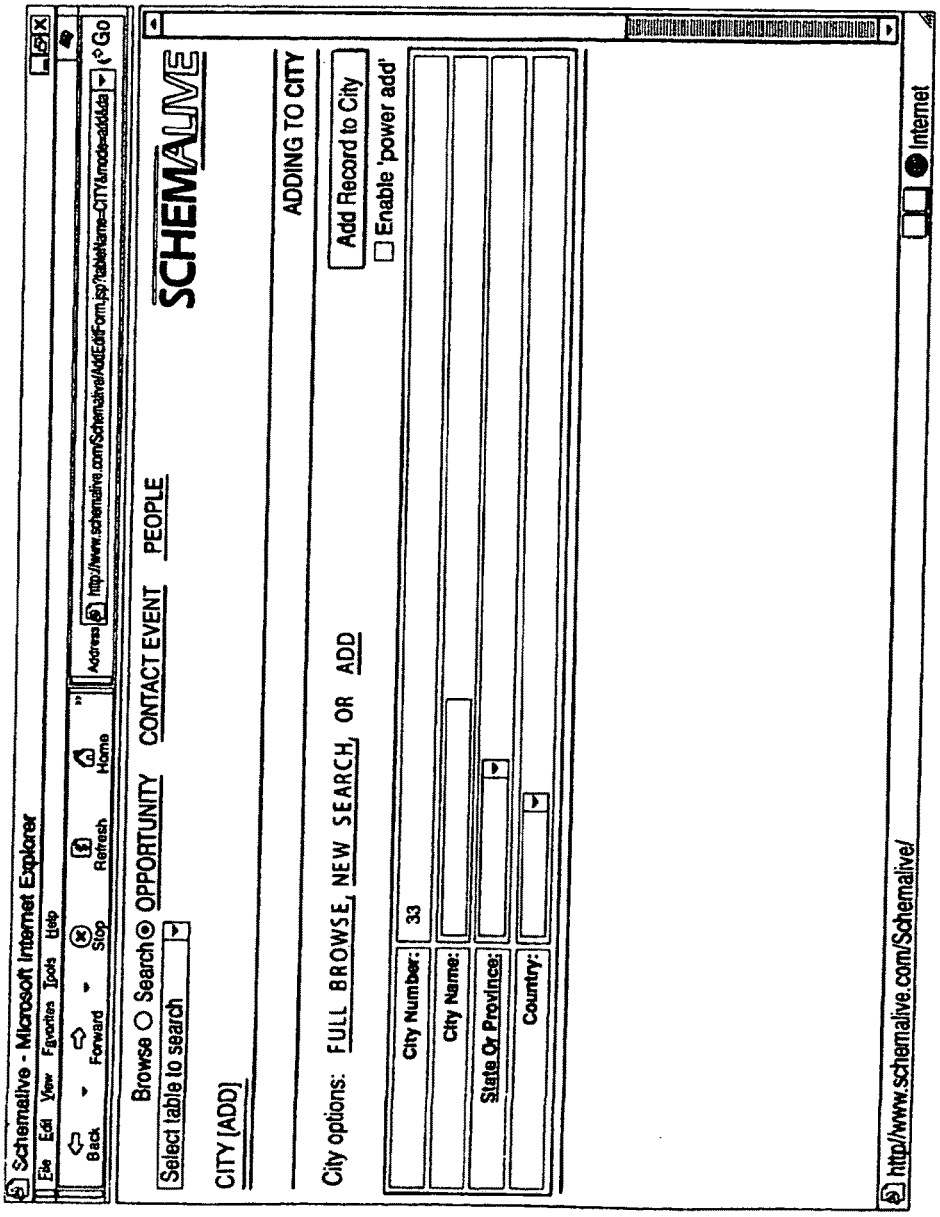

Schemalive - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://www.schemalive.com/Schemalive/AddEditForm.jsp?tableName=CITY&mode=add&da
Go
Browse Search OPPORTUNITY CONTACT EVENT PEOPLE
SCHEMALIVE
Select table to search
CITY [ADD]
ADDING TO CITY
City options: FULL BROWSE, NEW SEARCH, OR ADD
Add Record to City
Enable 'power add'
City Number: 33
City Name:
State Or Province:
Country:
http://www.schemalive.com/Schemalive/
Internet

FIG. 4

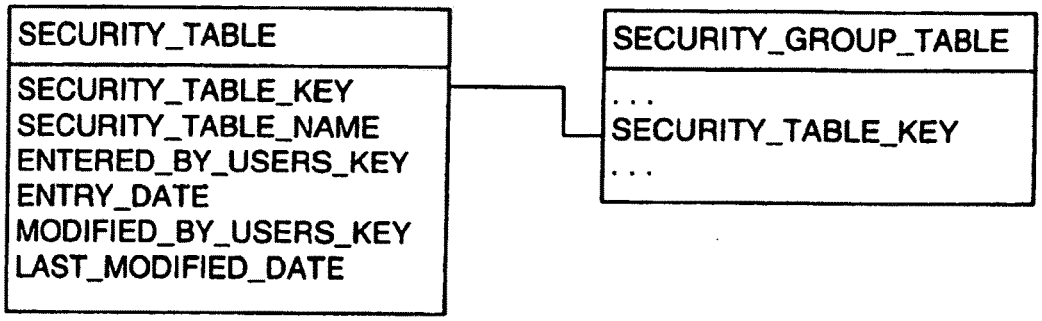

SECURITY_TABLE
SECURITY_TABLE_KEY
SECURITY_TABLE_NAME
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
SECURITY_GROUP_TABLE
. . .
SECURITY_TABLE_KEY
. . .

Fig. 5A

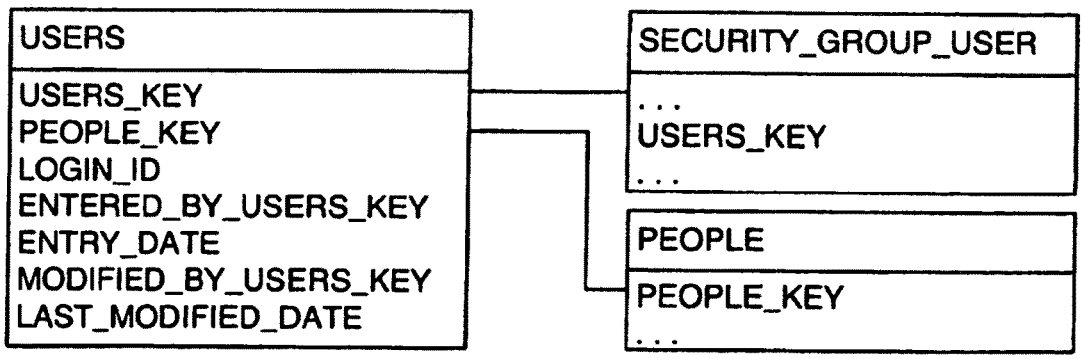

USERS
USERS_KEY
PEOPLE_KEY
LOGIN_ID
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
SECURITY_GROUP_USER
. . .
USERS_KEY
. . .
PEOPLE
PEOPLE_KEY
. . .

Fig. 5B

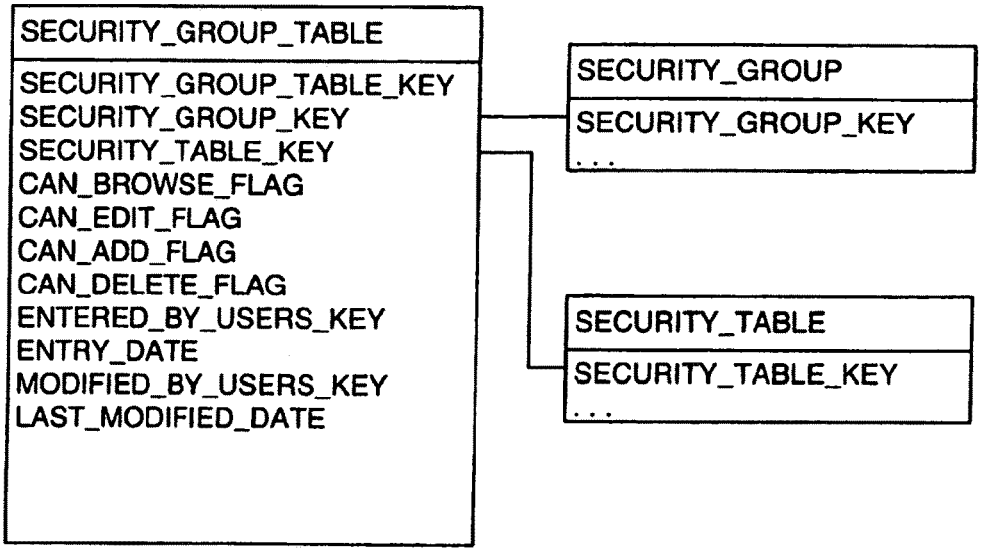

SECURITY_GROUP_TABLE
SECURITY_GROUP_TABLE_KEY
SECURITY_GROUP_KEY
SECURITY_TABLE_KEY
CAN_BROWSE_FLAG
CAN_EDIT_FLAG
CAN_ADD_FLAG
CAN_DELETE_FLAG
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
SECURITY_GROUP
SECURITY_GROUP_KEY
. . .
SECURITY_TABLE
SECURITY_TABLE_KEY
. . .

Fig. 5C

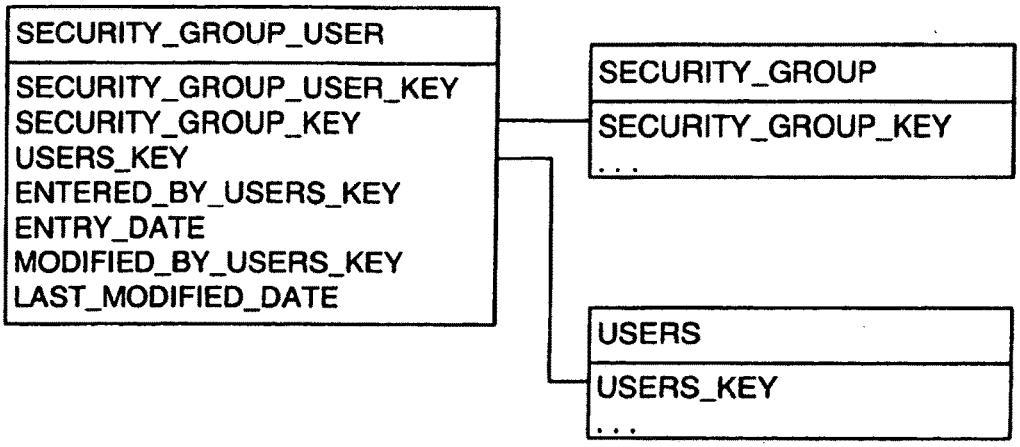

SECURITY_GROUP_USER
SECURITY_GROUP_USER_KEY
SECURITY_GROUP_KEY
USERS_KEY
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
SECURITY_GROUP
SECURITY_GROUP_KEY
. . .
USERS
USERS_KEY
. . .

Fig. 5D

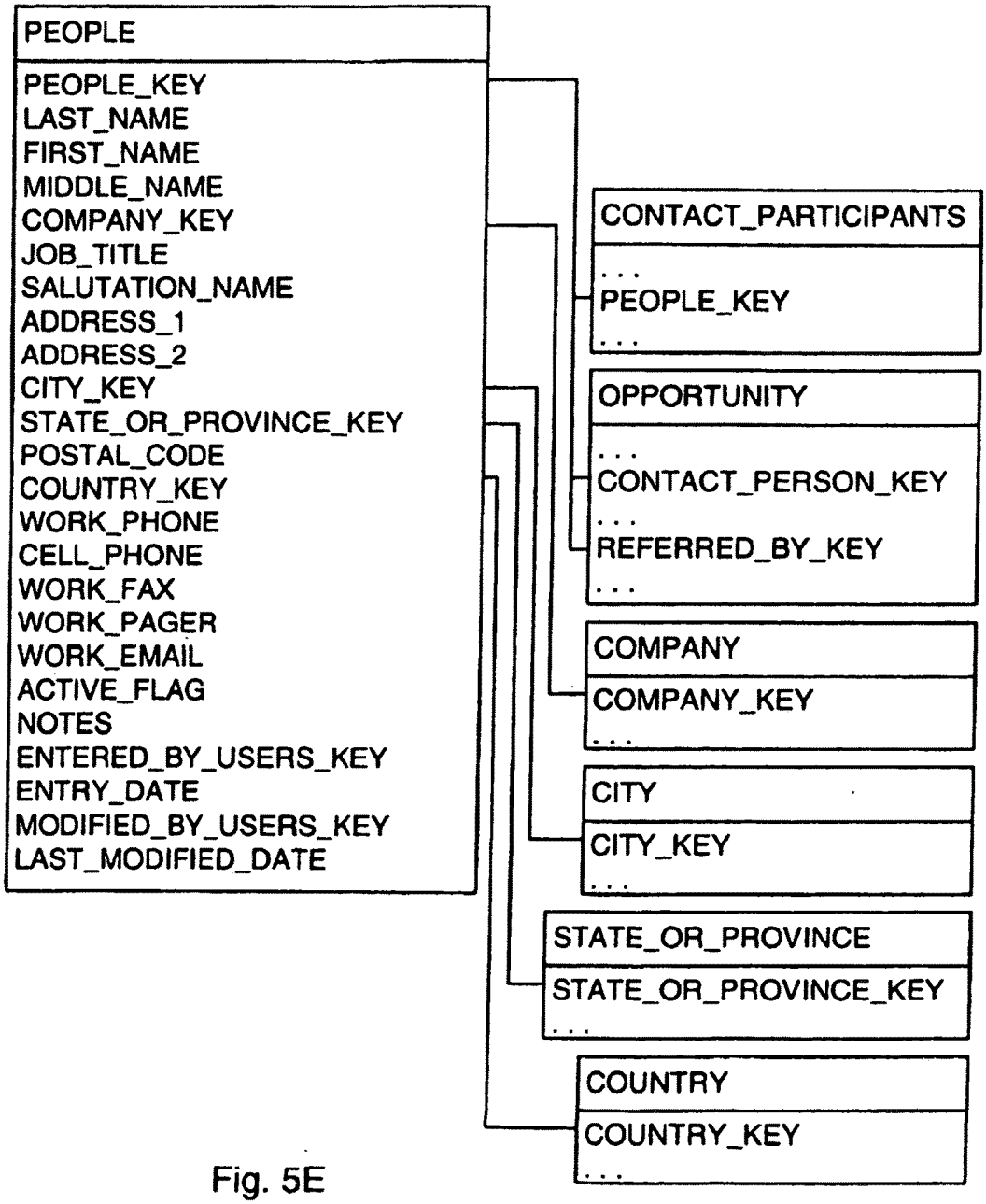

PEOPLE
PEOPLE_KEY
LAST_NAME
FIRST_NAME
MIDDLE_NAME
COMPANY_KEY
JOB_TITLE
SALUTATION_NAME
ADDRESS_1
ADDRESS_2
CITY_KEY
STATE_OR_PROVINCE_KEY
POSTAL_CODE
COUNTRY_KEY
WORK_PHONE
CELL_PHONE
WORK_FAX
WORK_PAGER
WORK_EMAIL
ACTIVE_FLAG
NOTES
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
CONTACT_PARTICIPANTS
. . .
PEOPLE_KEY
. . .
OPPORTUNITY
. . .
CONTACT_PERSON_KEY
. . .
REFERRED_BY_KEY
. . .
COMPANY
COMPANY_KEY
. . .
CITY
CITY_KEY
. . .
STATE_OR_PROVINCE
STATE_OR_PROVINCE_KEY
. . .
COUNTRY
COUNTRY_KEY
. . .

Fig. 5E

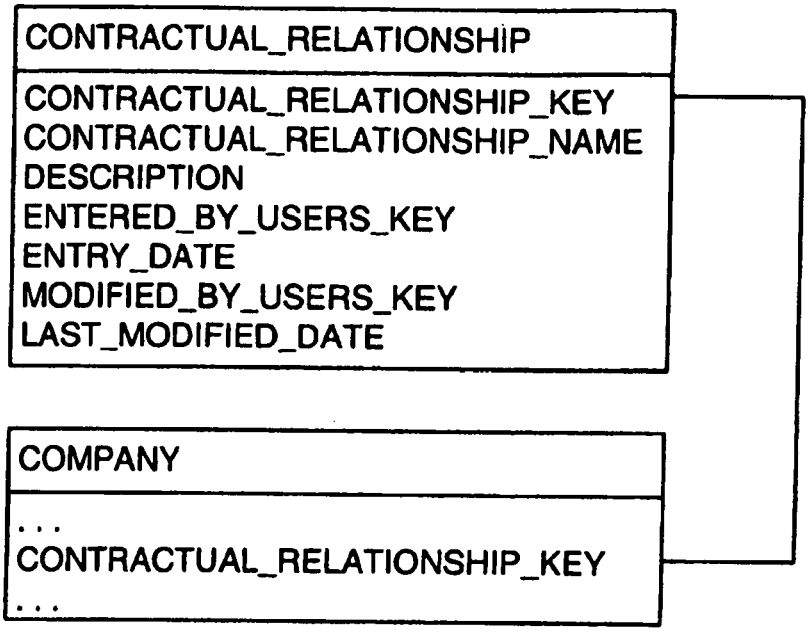

CONTRACTUAL_RELATIONSHIP
CONTRACTUAL_RELATIONSHIP_KEY
CONTRACTUAL_RELATIONSHIP_NAME
DESCRIPTION
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
COMPANY
. . .
CONTRACTUAL_RELATIONSHIP_KEY
. . .

Fig. 5F

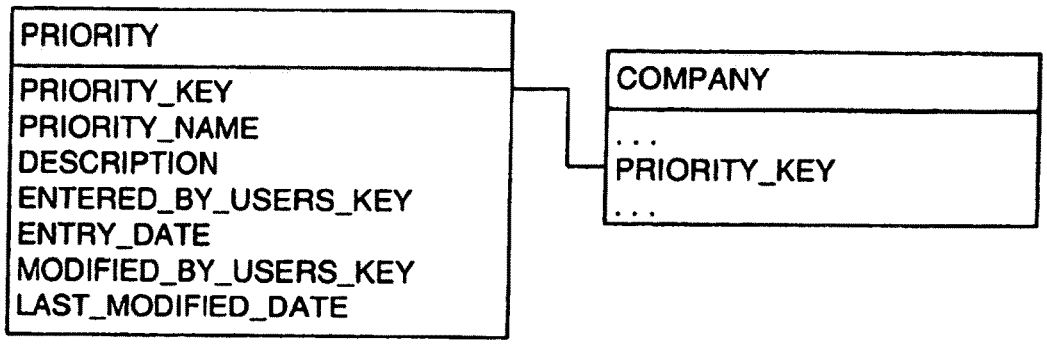

PRIORITY
PRIORITY_KEY
PRIORITY_NAME
DESCRIPTION
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
COMPANY
. . .
PRIORITY_KEY
. . .

Fig. 5G

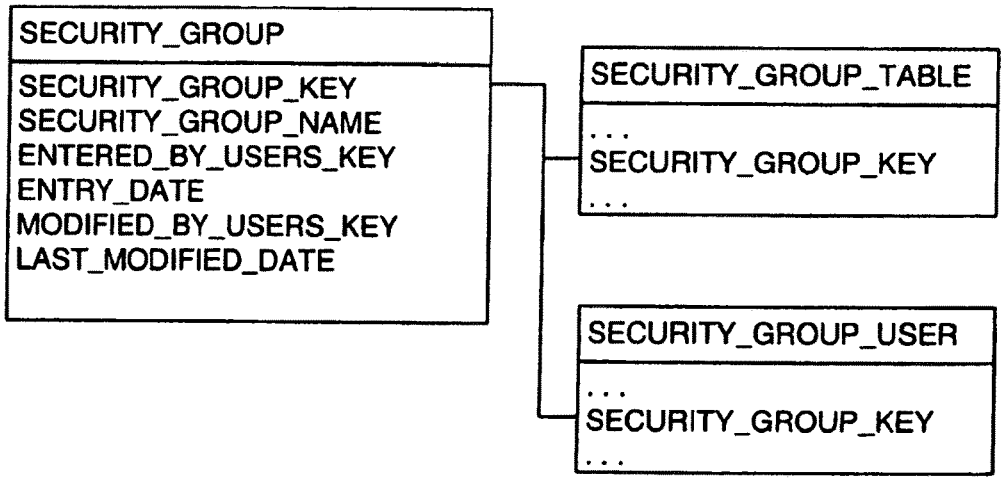

SECURITY_GROUP
SECURITY_GROUP_KEY
SECURITY_GROUP_NAME
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
SECURITY_GROUP_TABLE
. . .
SECURITY_GROUP_KEY
. . .
SECURITY_GROUP_USER
. . .
SECURITY_GROUP_KEY
. . .

Fig. 5H

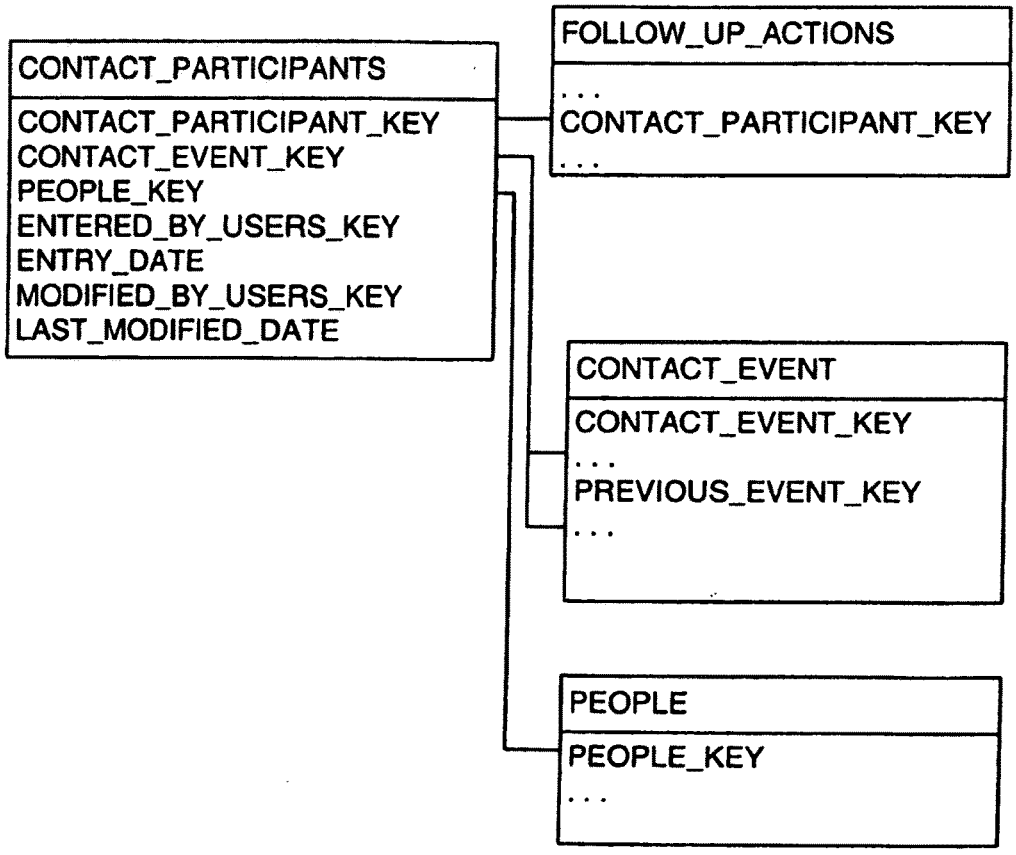

CONTACT_PARTICIPANTS
CONTACT_PARTICIPANT_KEY
CONTACT_EVENT_KEY
PEOPLE_KEY
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
FOLLOW_UP_ACTIONS
. . .
CONTACT_PARTICIPANT_KEY
. . .
CONTACT_EVENT
CONTACT_EVENT_KEY
. . .
PREVIOUS_EVENT_KEY
. . .
PEOPLE
PEOPLE_KEY
. . .

Fig. 5I

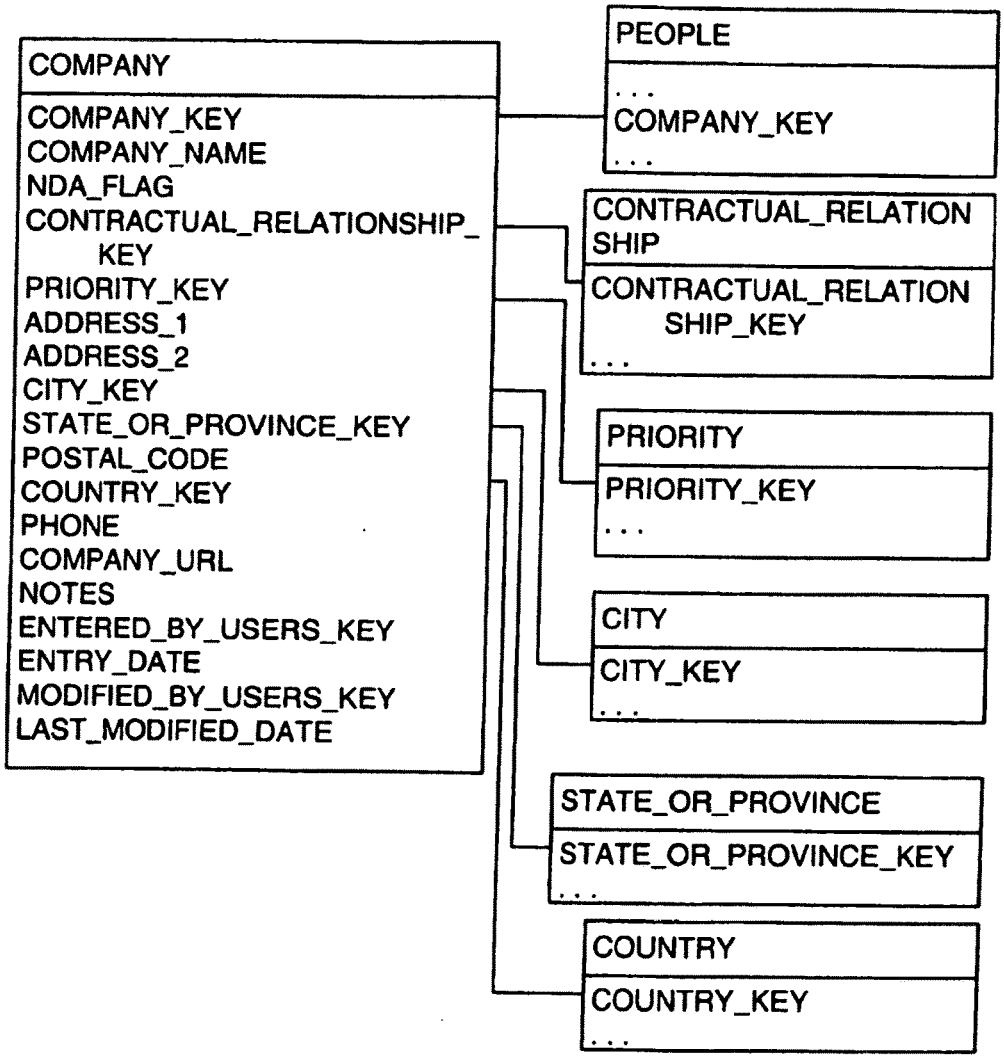

COMPANY
COMPANY_KEY
COMPANY_NAME
NDA_FLAG
CONTRACTUAL_RELATIONSHIP_KEY
PRIORITY_KEY
ADDRESS_1
ADDRESS_2
CITY_KEY
STATE_OR_PROVINCE_KEY
POSTAL_CODE
COUNTRY_KEY
PHONE
COMPANY_URL
NOTES
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
PEOPLE
. . .
COMPANY_KEY
. . .
CONTRACTUAL_RELATIONSHIP
CONTRACTUAL_RELATIONSHIP_KEY
. . .
PRIORITY
PRIORITY_KEY
. . .
CITY
CITY_KEY
. . .
STATE_OR_PROVINCE
STATE_OR_PROVINCE_KEY
. . .
COUNTRY
COUNTRY_KEY
. . .

Fig. 5J

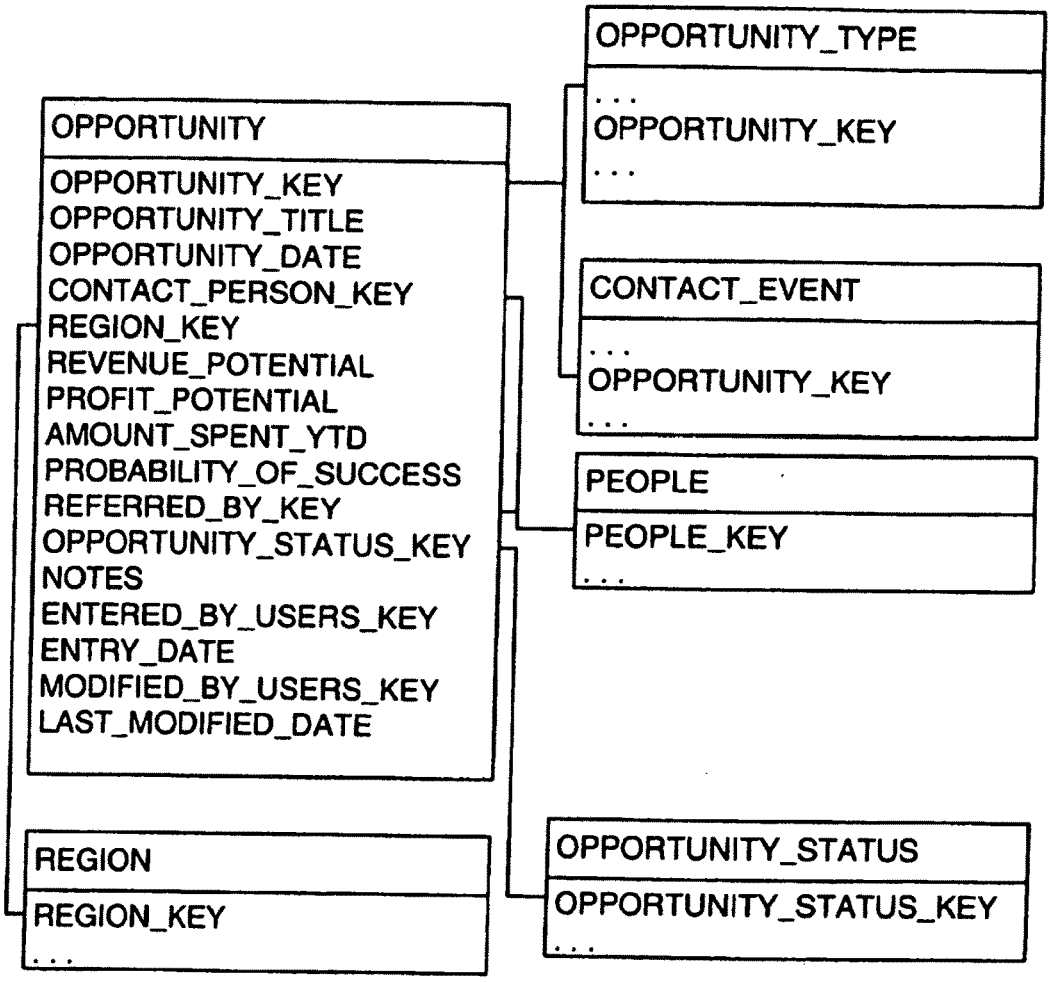

OPPORTUNITY_TYPE
. . .
OPPORTUNITY_KEY
. . .
OPPORTUNITY
OPPORTUNITY_KEY
OPPORTUNITY_TITLE
OPPORTUNITY_DATE
CONTACT_PERSON_KEY
REGION_KEY
REVENUE_POTENTIAL
PROFIT_POTENTIAL
AMOUNT_SPENT_YTD
PROBABILITY_OF_SUCCESS
REFERRED_BY_KEY
OPPORTUNITY_STATUS_KEY
NOTES
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
CONTACT_EVENT
. . .
OPPORTUNITY_KEY
. . .
PEOPLE
PEOPLE_KEY
. . .
REGION
REGION_KEY
. . .
OPPORTUNITY_STATUS
OPPORTUNITY_STATUS_KEY
. . .

Fig. 5K

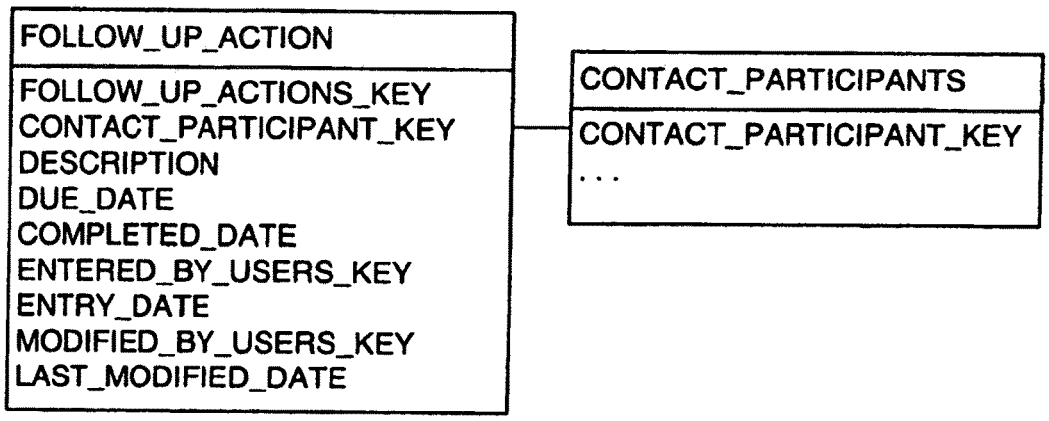

FOLLOW_UP_ACTION
FOLLOW_UP_ACTIONS_KEY
CONTACT_PARTICIPANT_KEY
DESCRIPTION
DUE_DATE
COMPLETED_DATE
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
CONTACT_PARTICIPANTS
CONTACT_PARTICIPANT_KEY
. . .

Fig. 5L

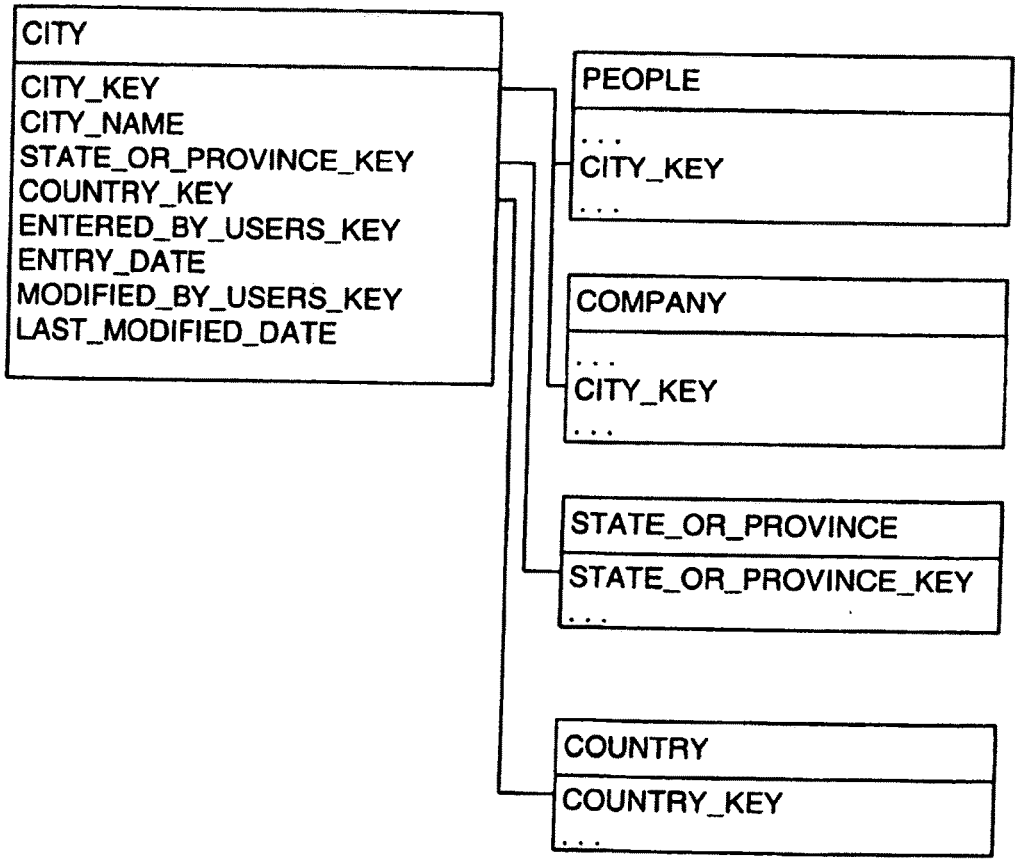

CITY
CITY_KEY
CITY_NAME
STATE_OR_PROVINCE_KEY
COUNTRY_KEY
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
PEOPLE
. . .
CITY_KEY
. . .
COMPANY
. . .
CITY_KEY
. . .
STATE_OR_PROVINCE
STATE_OR_PROVINCE_KEY
. . .
COUNTRY
COUNTRY_KEY
. . .

Fig. 5M

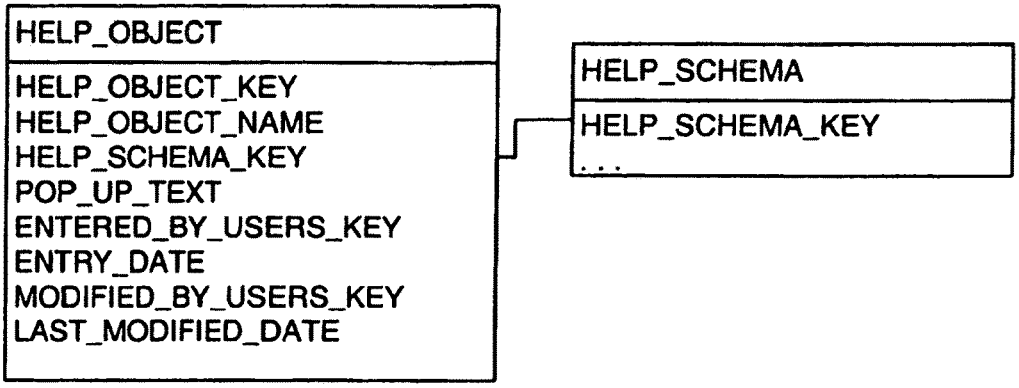

HELP_OBJECT
HELP_OBJECT_KEY
HELP_OBJECT_NAME
HELP_SCHEMA_KEY
POP_UP_TEXT
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
HELP_SCHEMA
HELP_SCHEMA_KEY
. . .

Fig . 5 N

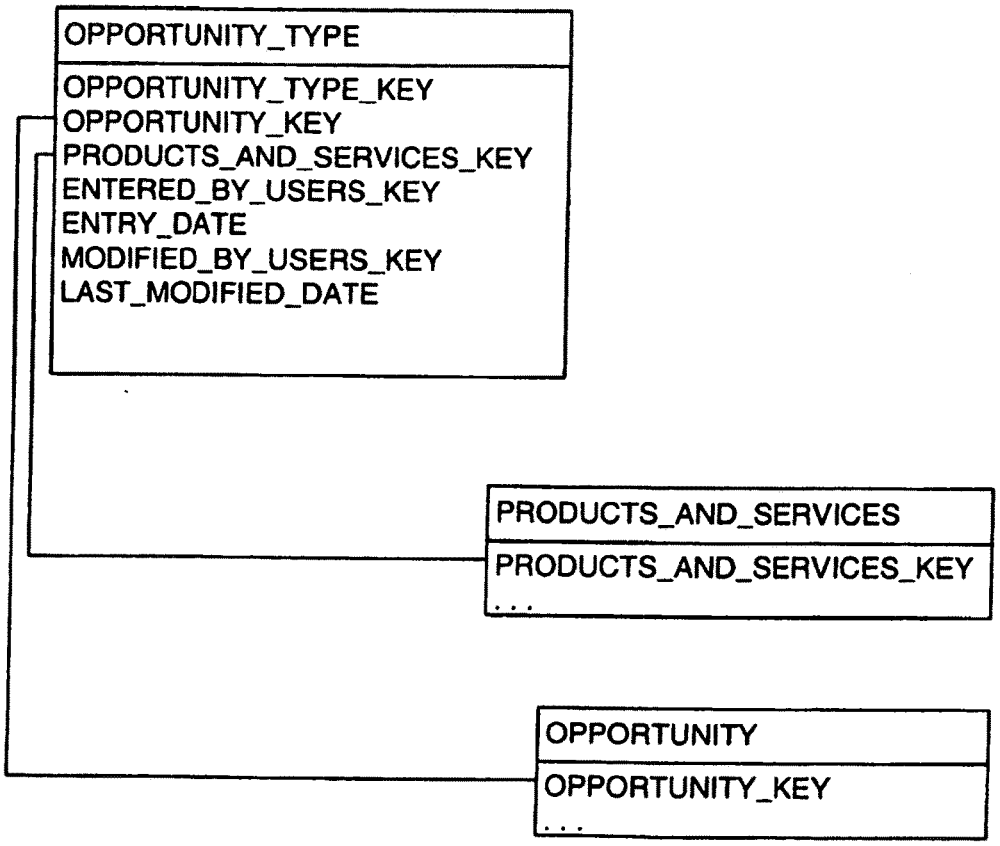

OPPORTUNITY_TYPE
OPPORTUNITY_TYPE_KEY
OPPORTUNITY_KEY
PRODUCTS_AND_SERVICES_KEY
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
PRODUCTS_AND_SERVICES
PRODUCTS_AND_SERVICES_KEY
. . .
OPPORTUNITY
OPPORTUNITY_KEY
. . .

Fig. 5O

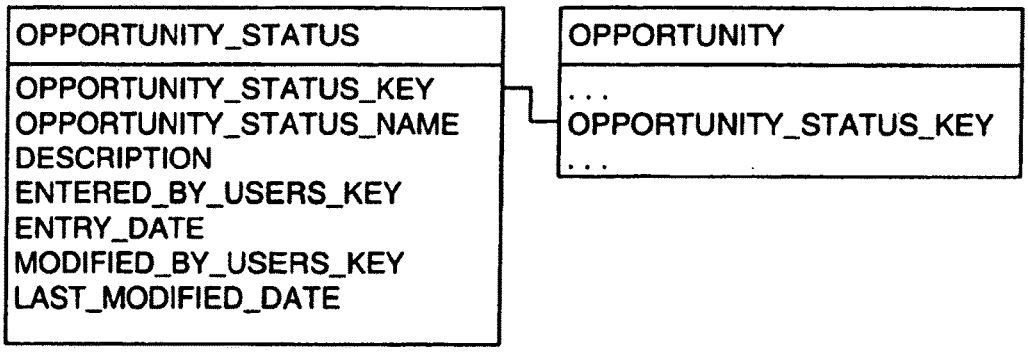

OPPORTUNITY_STATUS
OPPORTUNITY_STATUS_KEY
OPPORTUNITY_STATUS_NAME
DESCRIPTION
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
OPPORTUNITY
. . .
OPPORTUNITY_STATUS_KEY
. . .

Fig. 5P

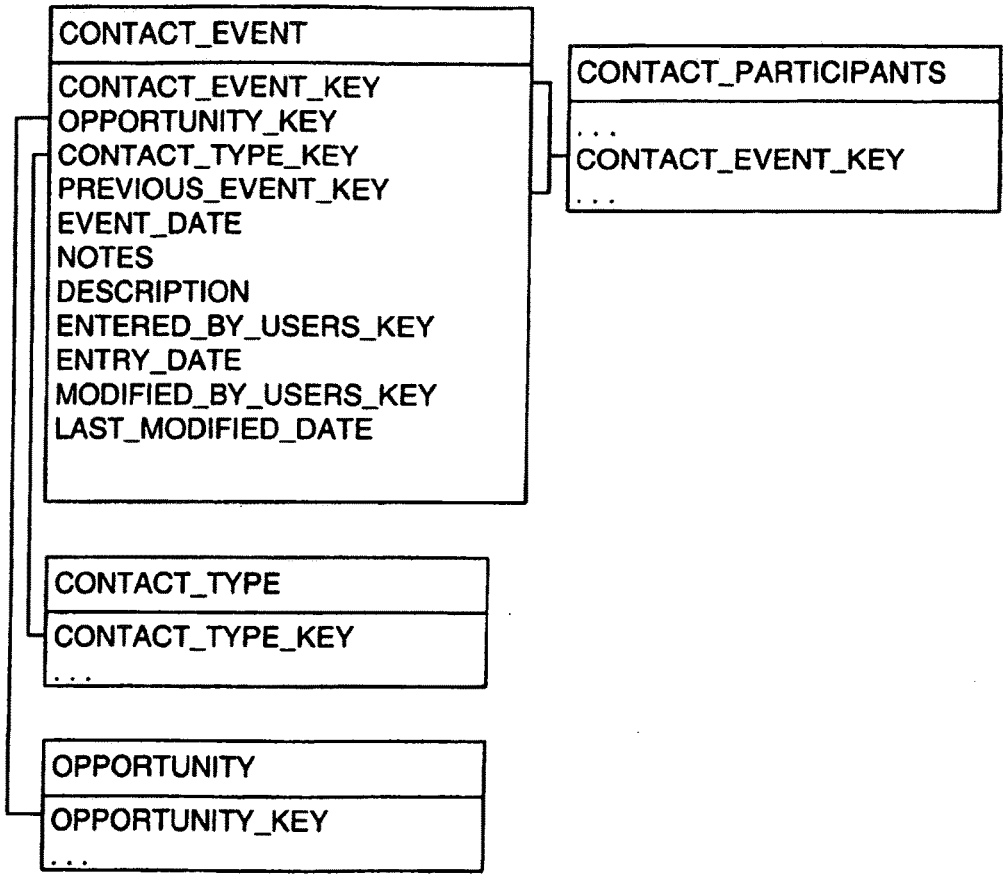

CONTACT_EVENT
CONTACT_EVENT_KEY
OPPORTUNITY_KEY
CONTACT_TYPE_KEY
PREVIOUS_EVENT_KEY
EVENT_DATE
NOTES
DESCRIPTION
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
CONTACT_PARTICIPANTS
. . .
CONTACT_EVENT_KEY
. . .
CONTACT_TYPE
CONTACT_TYPE_KEY
. . .
OPPORTUNITY
OPPORTUNITY_KEY
. . .

Fig. 5Q

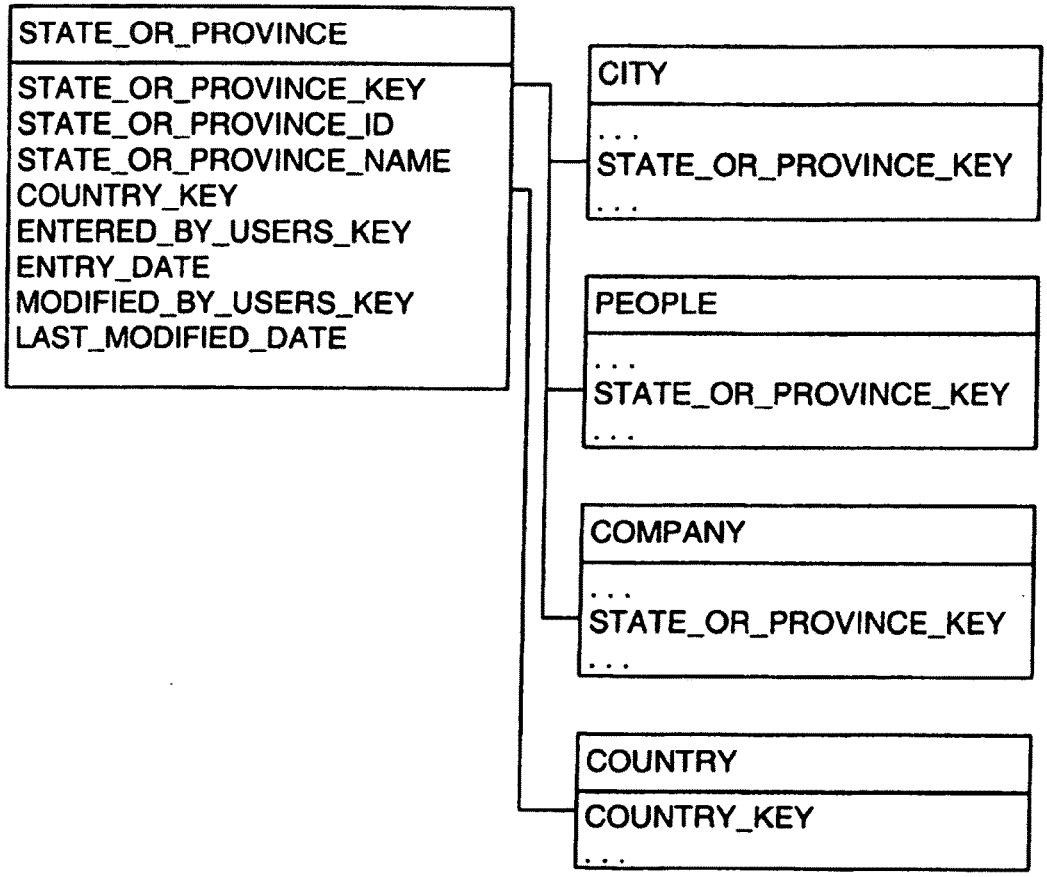

STATE_OR_PROVINCE
STATE_OR_PROVINCE_KEY
STATE_OR_PROVINCE_ID
STATE_OR_PROVINCE_NAME
COUNTRY_KEY
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
CITY
. . .
STATE_OR_PROVINCE_KEY
. . .
PEOPLE
. . .
STATE_OR_PROVINCE_KEY
. . .
COMPANY
. . .
STATE_OR_PROVINCE_KEY
. . .
COUNTRY
COUNTRY_KEY
. . .

Fig. 5R

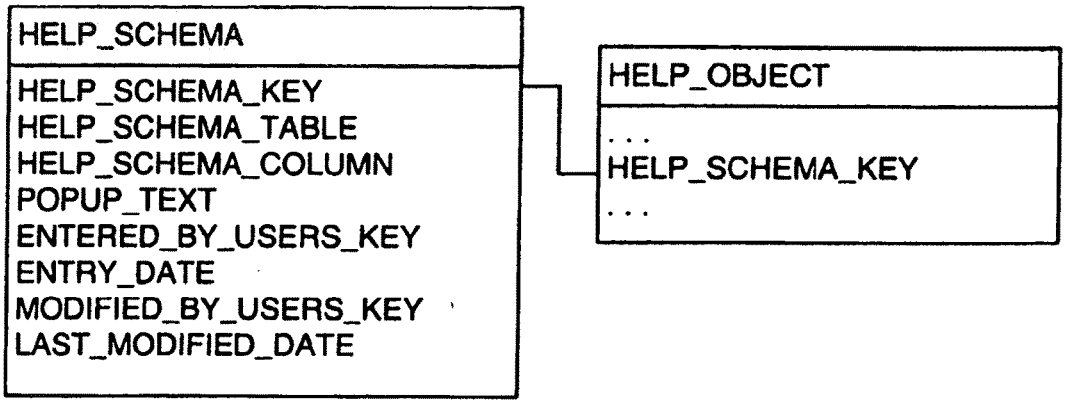

HELP_SCHEMA
HELP_SCHEMA_KEY
HELP_SCHEMA_TABLE
HELP_SCHEMA_COLUMN
POPUP_TEXT
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
HELP_OBJECT
. . .
HELP_SCHEMA_KEY
. . .

Fig. 5S

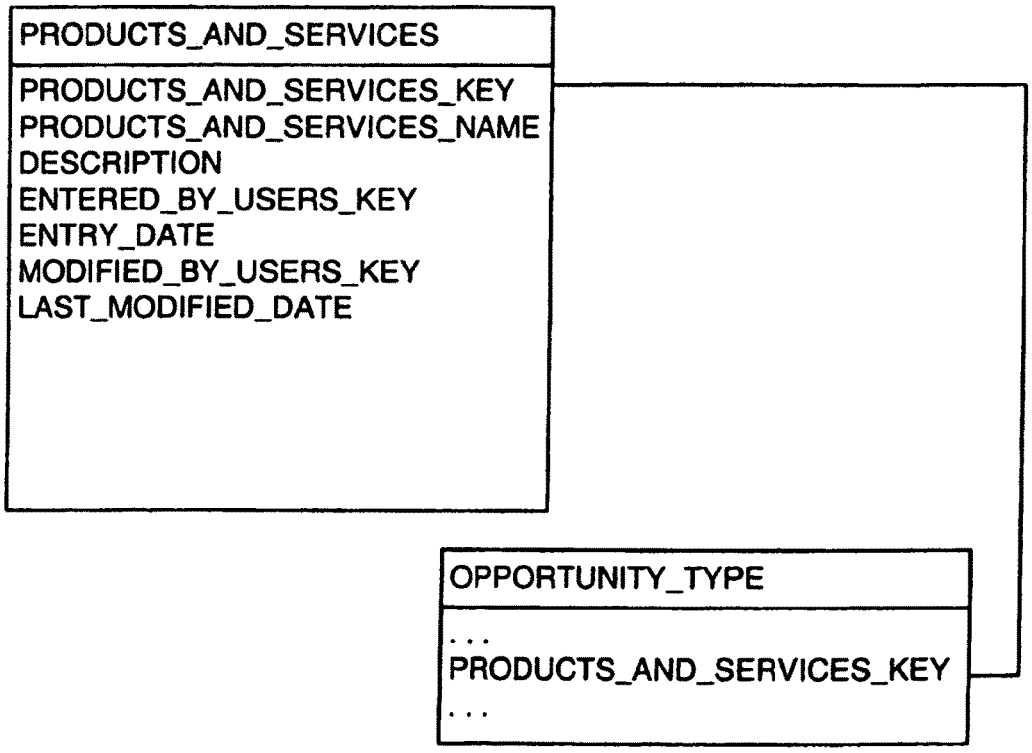

PRODUCTS_AND_SERVICES
PRODUCTS_AND_SERVICES_KEY
PRODUCTS_AND_SERVICES_NAME
DESCRIPTION
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
OPPORTUNITY_TYPE
. . .
PRODUCTS_AND_SERVICES_KEY
. . .

Fig. 5T

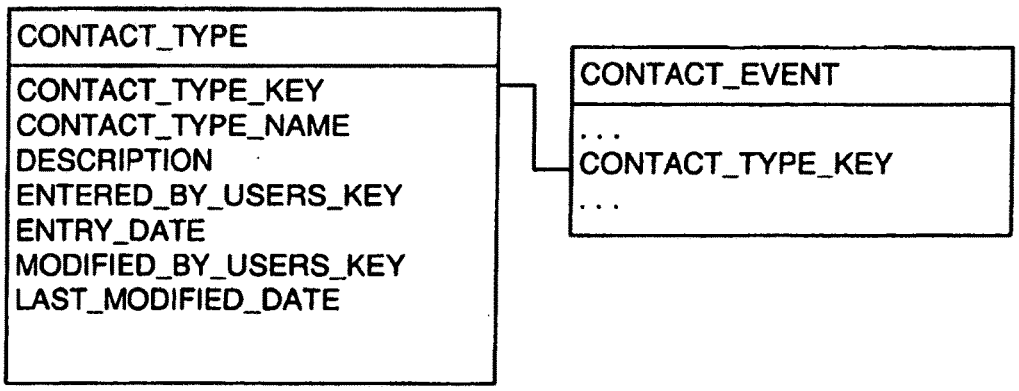

CONTACT_TYPE
CONTACT_TYPE_KEY
CONTACT_TYPE_NAME
DESCRIPTION
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
CONTACT_EVENT
. . .
CONTACT_TYPE_KEY
. . .

Fig. 5U

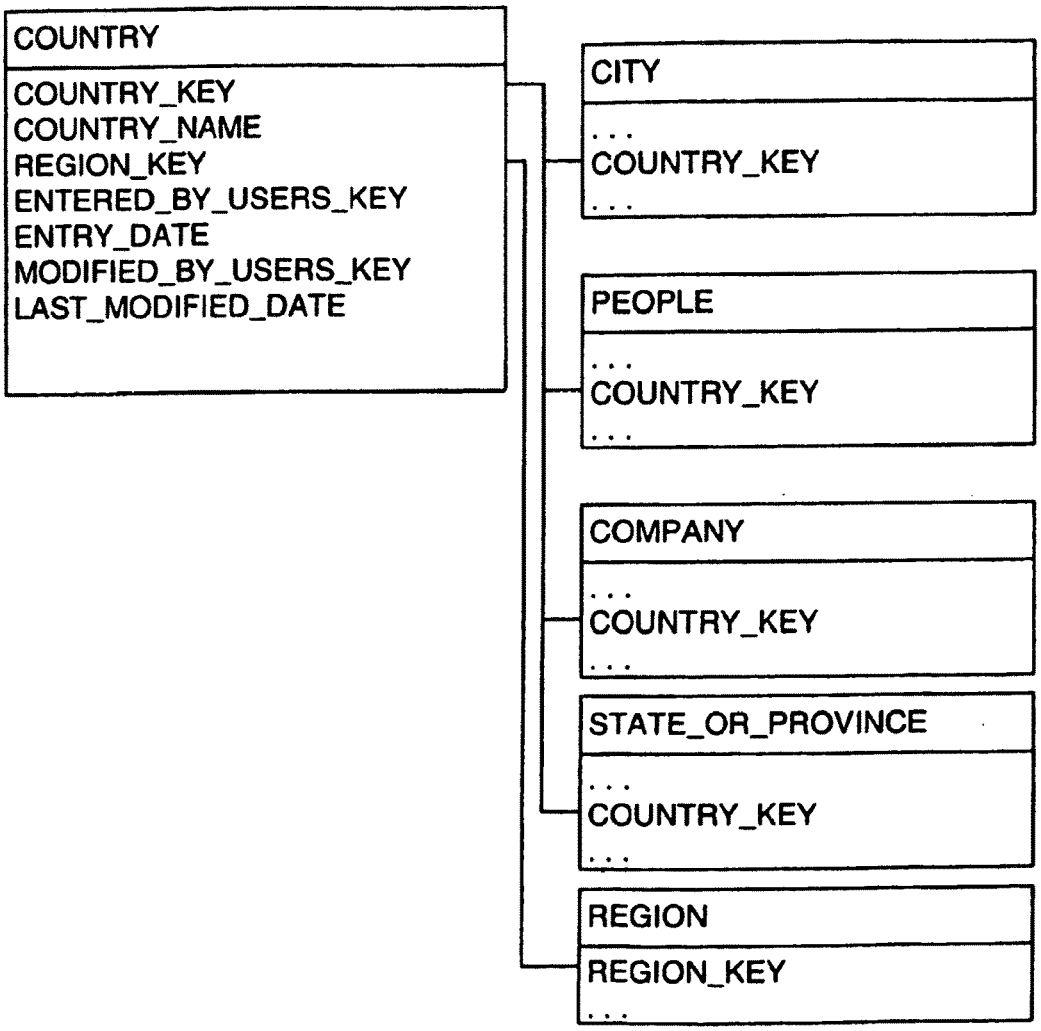

COUNTRY
COUNTRY_KEY
COUNTRY_NAME
REGION_KEY
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
CITY
. . .
COUNTRY_KEY
. . .
PEOPLE
. . .
COUNTRY_KEY
. . .
COMPANY
. . .
COUNTRY_KEY
. . .
STATE_OR_PROVINCE
. . .
COUNTRY_KEY
. . .
REGION
REGION_KEY
. . .

Fig. 5V

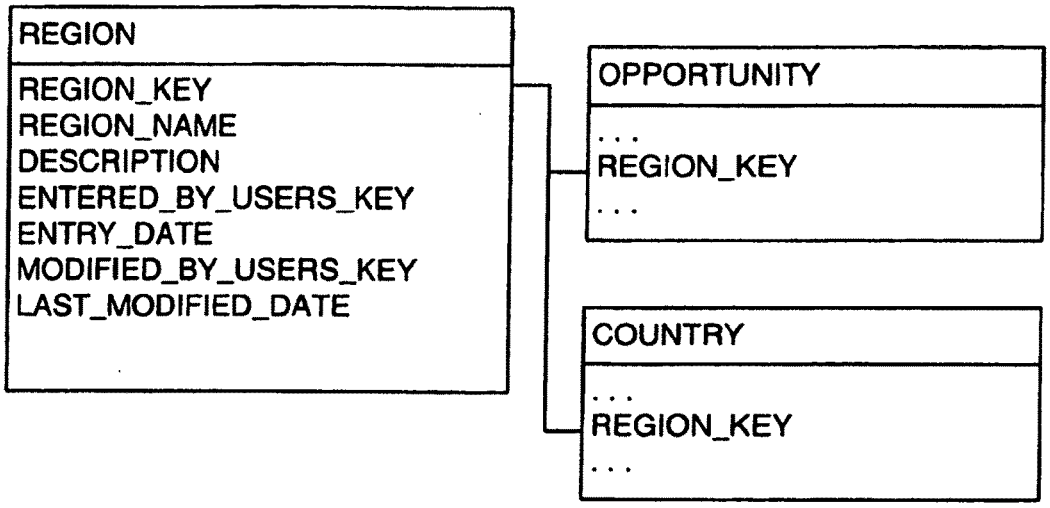

REGION
REGION_KEY
REGION_NAME
DESCRIPTION
ENTERED_BY_USERS_KEY
ENTRY_DATE
MODIFIED_BY_USERS_KEY
LAST_MODIFIED_DATE
OPPORTUNITY
. . .
REGION_KEY
. . .
COUNTRY
. . .
REGION_KEY
. . .

Fig. 5W

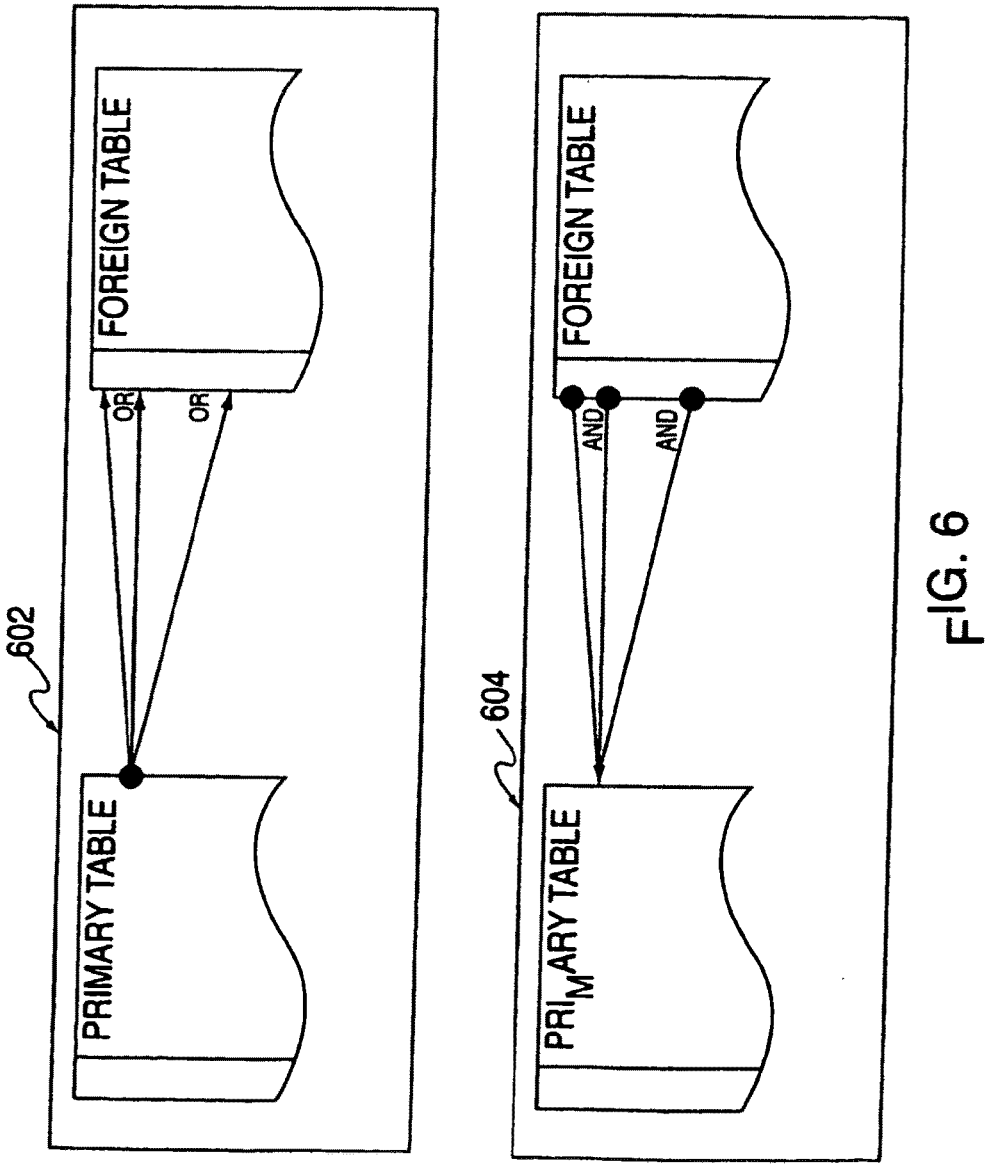

602
FOREIGN TABLE
OR
OR
PRIMARY TABLE
604
FOREIGN TABLE
AND
AND
PRIMARY TABLE

FIG. 6

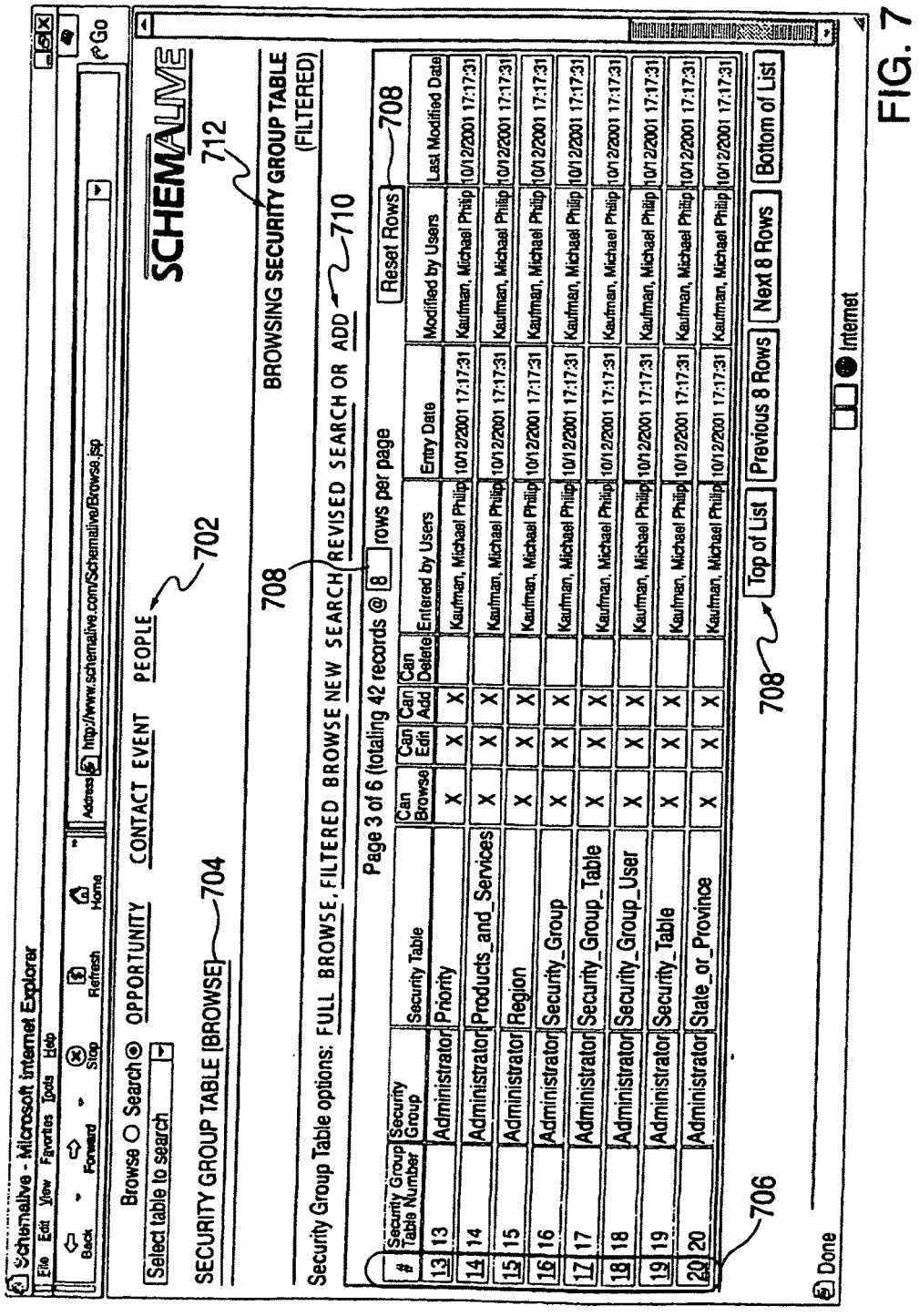

Schemalive - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://www.schemalive.com/Schemalive/Browse.jsp
Go
Browse Search OPPORTUNITY CONTACT EVENT PEOPLE
SCHEMALIVE
Select table to search
SECURITY GROUP TABLE [BROWSE]
704
702
712
708
BROWSING SECURITY GROUP TABLE (FILTERED)
Security Group Table options: FULL BROWSE, FILTERED BROWSE NEW SEARCH REVISED SEARCH OR ADD
710
Page 3 of 6 (totaling 42 records @ 8 rows per page
Reset Rows
708
| Security Group Table Number | Security Group | Security Table | Can Browse | Can Edit | Can Add | Can Delete | Entered by Users | Entry Date | Modified by Users | Last Modified Date
13 | 13 | Administrator | Priority | X | X | X | | Kaufman, Michael Philip | 10/12/2001 17:17:31 | Kaufman, Michael Philip | 10/12/2001 17:17:31
14 | 14 | Administrator | Products_and_Services | X | X | X | | Kaufman, Michael Philip | 10/12/2001 17:17:31 | Kaufman, Michael Philip | 10/12/2001 17:17:31
15 | 15 | Administrator | Region | X | X | X | | Kaufman, Michael Philip | 10/12/2001 17:17:31 | Kaufman, Michael Philip | 10/12/2001 17:17:31
16 | 16 | Administrator | Security_Group | X | X | X | | Kaufman, Michael Philip | 10/12/2001 17:17:31 | Kaufman, Michael Philip | 10/12/2001 17:17:31
17 | 17 | Administrator | Security_Group_Table | X | X | X | | Kaufman, Michael Philip | 10/12/2001 17:17:31 | Kaufman, Michael Philip | 10/12/2001 17:17:31
18 | 18 | Administrator | Security_Group_User | X | X | X | | Kaufman, Michael Philip | 10/12/2001 17:17:31 | Kaufman, Michael Philip | 10/12/2001 17:17:31
19 | 19 | Administrator | Security_Table | X | X | X | | Kaufman, Michael Philip | 10/12/2001 17:17:31 | Kaufman, Michael Philip | 10/12/2001 17:17:31
20 | 20 | Administrator | State_or_Province | X | X | X | | Kaufman, Michael Philip | 10/12/2001 17:17:31 | Kaufman, Michael Philip | 10/12/2001 17:17:31
706
708
Top of List
Previous 8 Rows
Next 8 Rows
Bottom of List
Done
Internet

FIG. 7

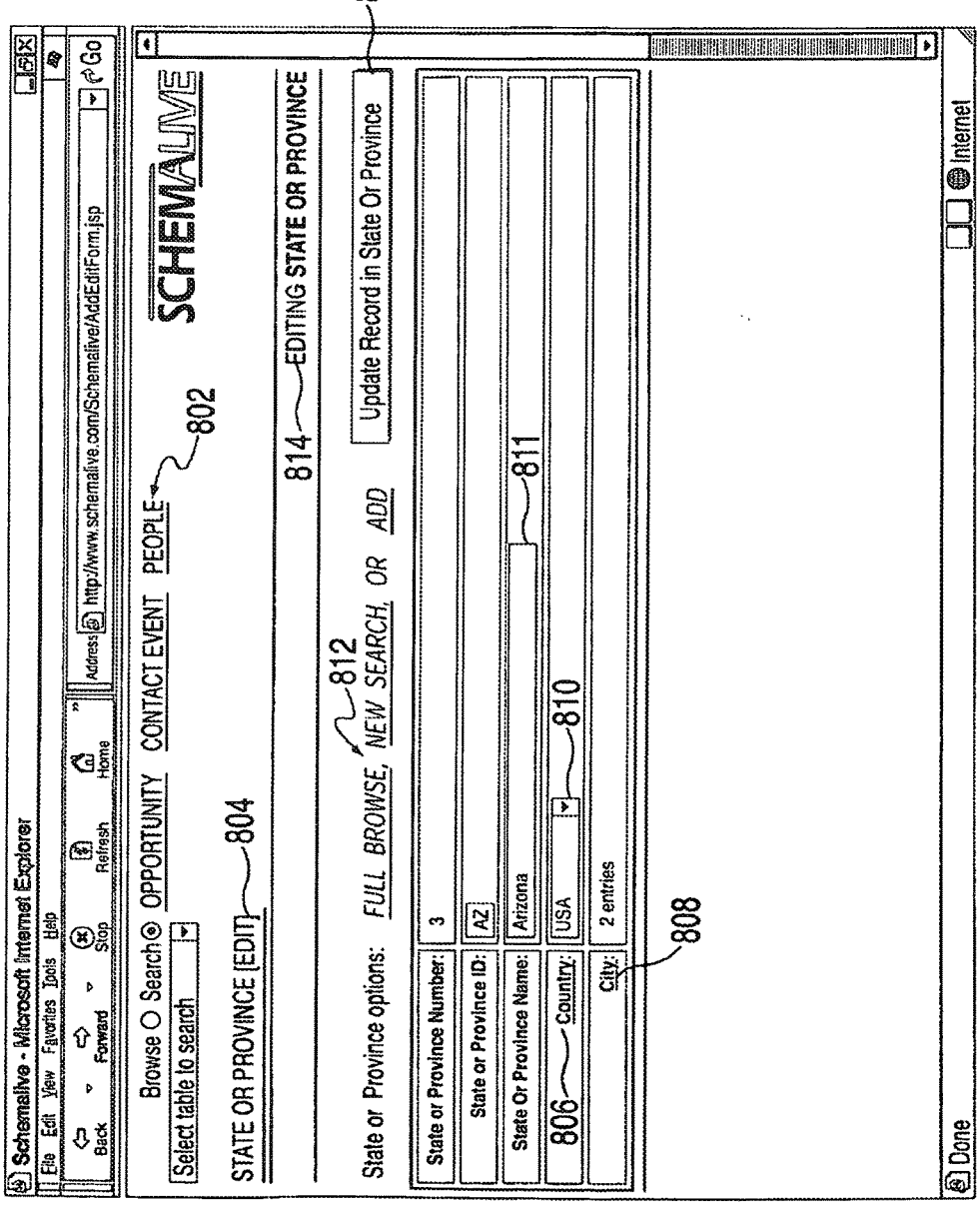

Schemalive - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://www.schemalive.com/Schemalive/AddEditForm.jsp
Go
Browse Search OPPORTUNITY CONTACT EVENT PEOPLE
802
SCHEMALIVE
Select table to search
STATE OR PROVINCE [EDIT]
804
814 EDITING STATE OR PROVINCE
816
812
State or Province options: FULL BROWSE, NEW SEARCH, OR ADD
Update Record in State Or Province
State or Province Number: 3
State or Province ID: AZ
State Or Province Name: Arizona
811
806 Country: USA
810
City: 2 entries
808
Done
Internet

FIG. 8

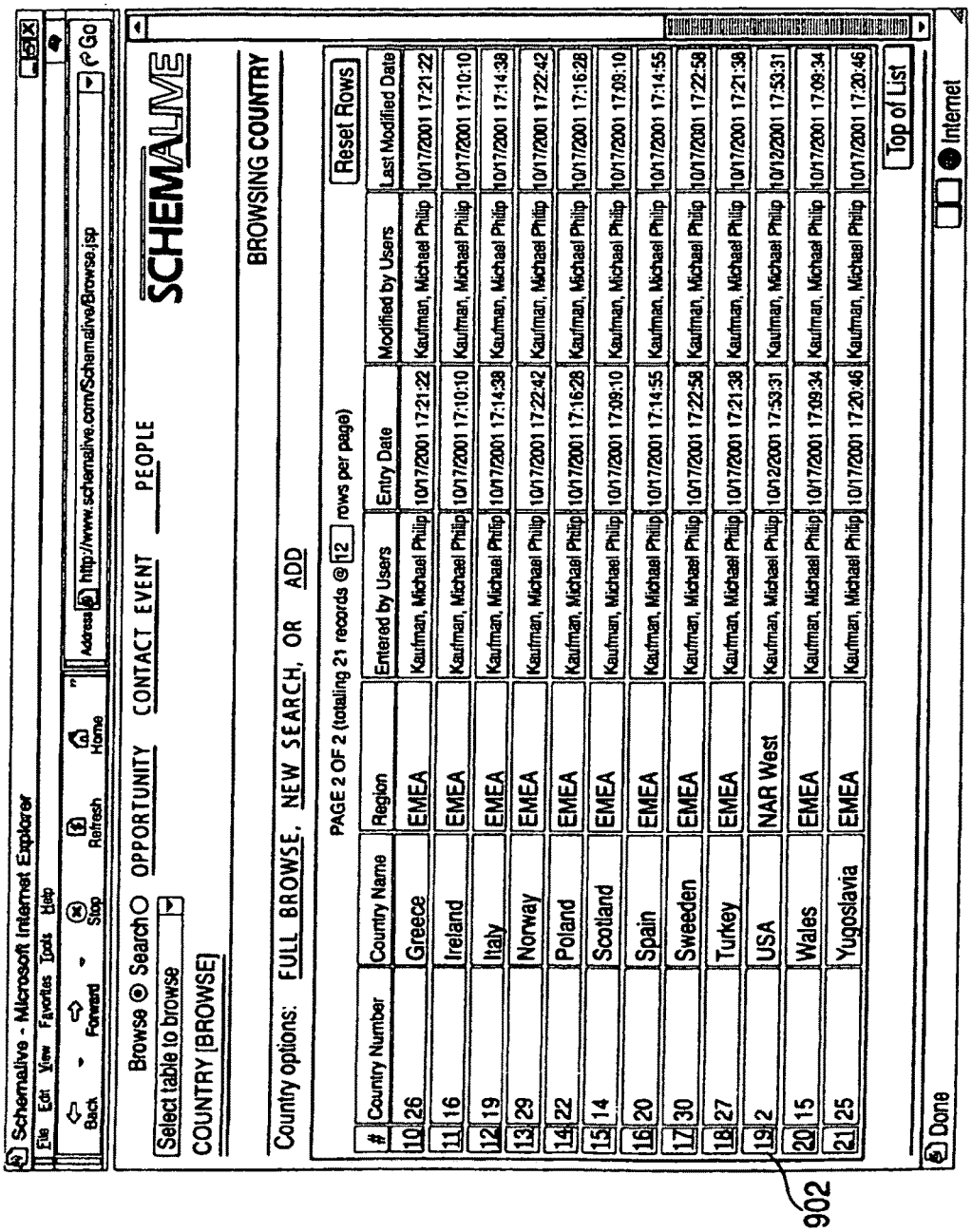

Schemalive - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://www.schemalive.com/Schemalive/Browse.jsp
Go
Browse Search OPPORTUNITY CONTACT EVENT PEOPLE
SCHEMALIVE
Select table to browse
COUNTRY [BROWSE]
BROWSING COUNTRY
Country options: FULL BROWSE, NEW SEARCH, OR ADD
PAGE 2 OF 2 (totaling 21 records @ 12 rows per page)
Reset Rows
Country Number Country Name Region Entered by Users Entry Date Modified by Users Last Modified Date
10 26 Greece EMEA Kaufman, Michael Philip 10/17/2001 17:21:22 Kaufman, Michael Philip 10/17/2001 17:21:22
11 16 Ireland EMEA Kaufman, Michael Philip 10/17/2001 17:10:10 Kaufman, Michael Philip 10/17/2001 17:10:10
12 19 Italy EMEA Kaufman, Michael Philip 10/17/2001 17:14:38 Kaufman, Michael Philip 10/17/2001 17:14:38
13 29 Norway EMEA Kaufman, Michael Philip 10/17/2001 17:22:42 Kaufman, Michael Philip 10/17/2001 17:22:42
14 22 Poland EMEA Kaufman, Michael Philip 10/17/2001 17:16:28 Kaufman, Michael Philip 10/17/2001 17:16:28
15 14 Scotland EMEA Kaufman, Michael Philip 10/17/2001 17:09:10 Kaufman, Michael Philip 10/17/2001 17:09:10
16 20 Spain EMEA Kaufman, Michael Philip 10/17/2001 17:14:55 Kaufman, Michael Philip 10/17/2001 17:14:55
17 30 Sweeden EMEA Kaufman, Michael Philip 10/17/2001 17:22:58 Kaufman, Michael Philip 10/17/2001 17:22:58
18 27 Turkey EMEA Kaufman, Michael Philip 10/17/2001 17:21:38 Kaufman, Michael Philip 10/17/2001 17:21:38
19 2 USA NAR West Kaufman, Michael Philip 10/12/2001 17:53:31 Kaufman, Michael Philip 10/12/2001 17:53:31
20 15 Wales EMEA Kaufman, Michael Philip 10/17/2001 17:09:34 Kaufman, Michael Philip 10/17/2001 17:09:34
21 25 Yugoslavia EMEA Kaufman, Michael Philip 10/17/2001 17:20:46 Kaufman, Michael Philip 10/17/2001 17:20:46
Top of List
Done
Internet
902

FIG. 9A

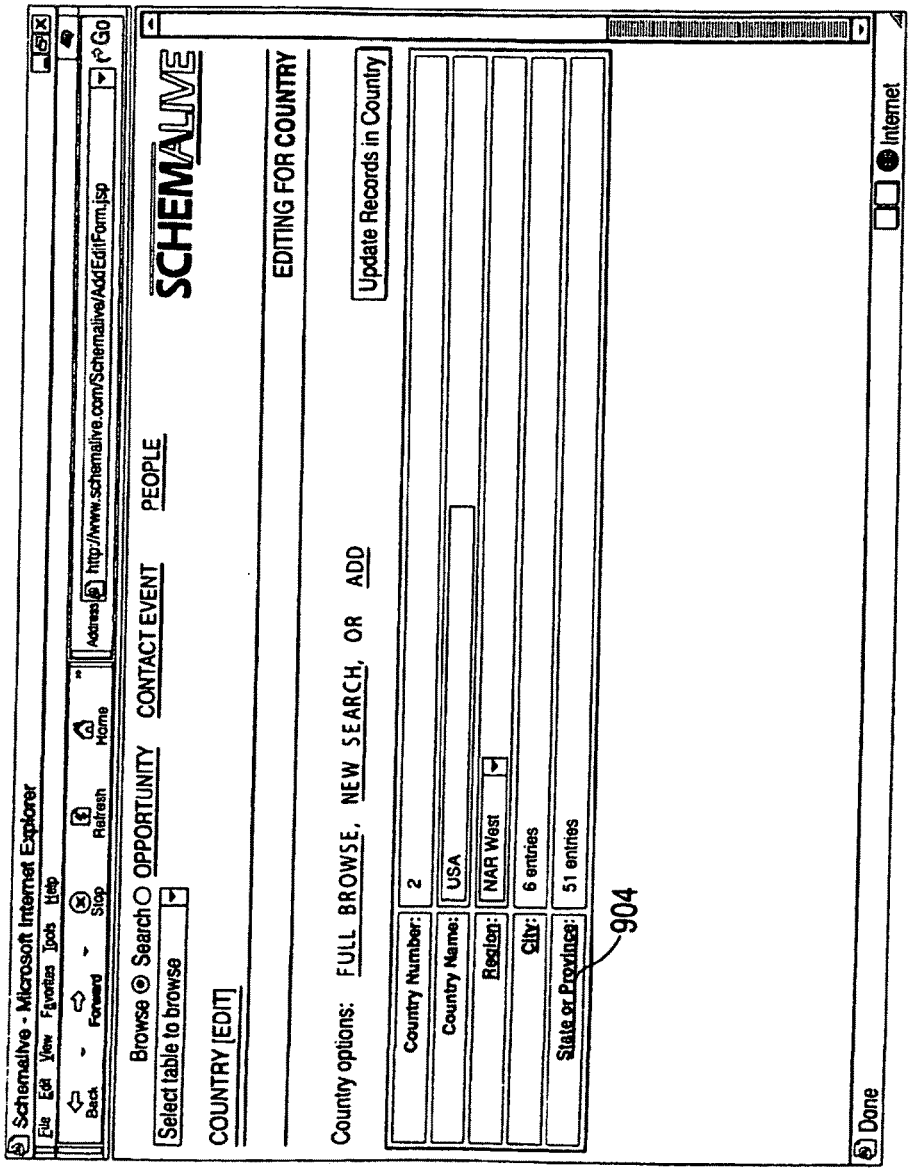

Schemalive - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://www.schemalive.com/Schemalive/AddEditForm.jsp
Go
Browse Search OPPORTUNITY CONTACT EVENT PEOPLE
SCHEMALIVE
Select table to browse
COUNTRY [EDIT]
EDITING FOR COUNTRY
Country options: FULL BROWSE, NEW SEARCH, OR ADD
Update Records in Country
Country Number: 2
Country Name: USA
Region: NAR West
City: 6 entries
State or Province: 51 entries
904
Done
Internet

FIG. 9B

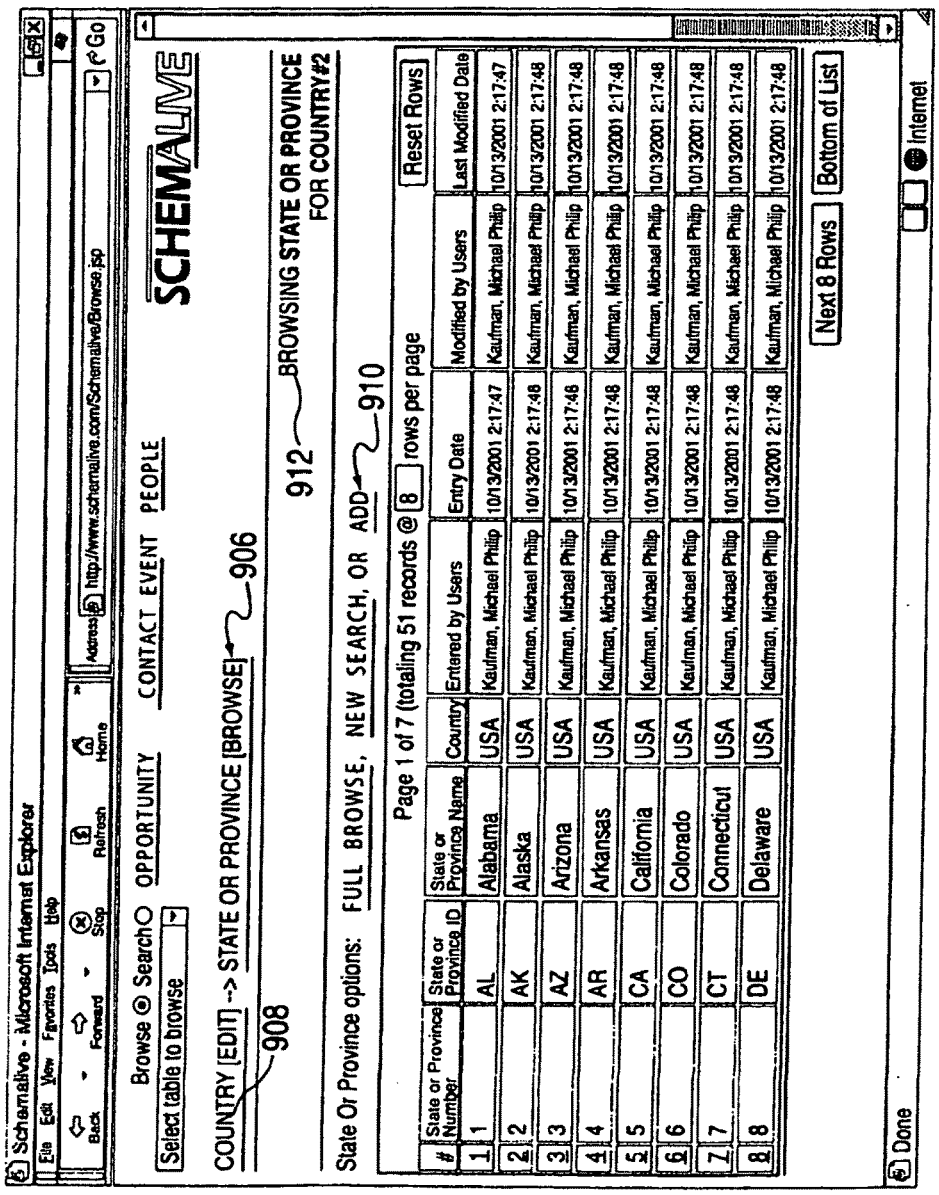

Schemalive - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://www.schemalive.com/Schemalive/Browse.jsp
Go
Browse Search OPPORTUNITY CONTACT EVENT PEOPLE
SCHEMALIVE
Select table to browse
COUNTRY [EDIT] -> STATE OR PROVINCE [BROWSE]
906
908
912
BROWSING STATE OR PROVINCE FOR COUNTRY#2
State Or Province options: FULL BROWSE, NEW SEARCH, OR ADD
910
Page 1 of 7 (totaling 51 records @ 8 rows per page
Reset Rows
| State or Province Number | State or Province ID | State or Province Name | Country | Entered by Users | Entry Date | Modified by Users | Last Modified Date
1 | 1 | AL | Alabama | USA | Kaufman, Michael Philip | 10/13/2001 2:17:47 | Kaufman, Michael Philip | 10/13/2001 2:17:47
2 | 2 | AK | Alaska | USA | Kaufman, Michael Philip | 10/13/2001 2:17:48 | Kaufman, Michael Philip | 10/13/2001 2:17:48
3 | 3 | AZ | Arizona | USA | Kaufman, Michael Philip | 10/13/2001 2:17:48 | Kaufman, Michael Philip | 10/13/2001 2:17:48
4 | 4 | AR | Arkansas | USA | Kaufman, Michael Philip | 10/13/2001 2:17:48 | Kaufman, Michael Philip | 10/13/2001 2:17:48
5 | 5 | CA | California | USA | Kaufman, Michael Philip | 10/13/2001 2:17:48 | Kaufman, Michael Philip | 10/13/2001 2:17:48
6 | 6 | CO | Colorado | USA | Kaufman, Michael Philip | 10/13/2001 2:17:48 | Kaufman, Michael Philip | 10/13/2001 2:17:48
7 | 7 | CT | Connecticut | USA | Kaufman, Michael Philip | 10/13/2001 2:17:48 | Kaufman, Michael Philip | 10/13/2001 2:17:48
8 | 8 | DE | Delaware | USA | Kaufman, Michael Philip | 10/13/2001 2:17:48 | Kaufman, Michael Philip | 10/13/2001 2:17:48
Next 8 Rows
Bottom of List
Done
Internet

FIG. 9C

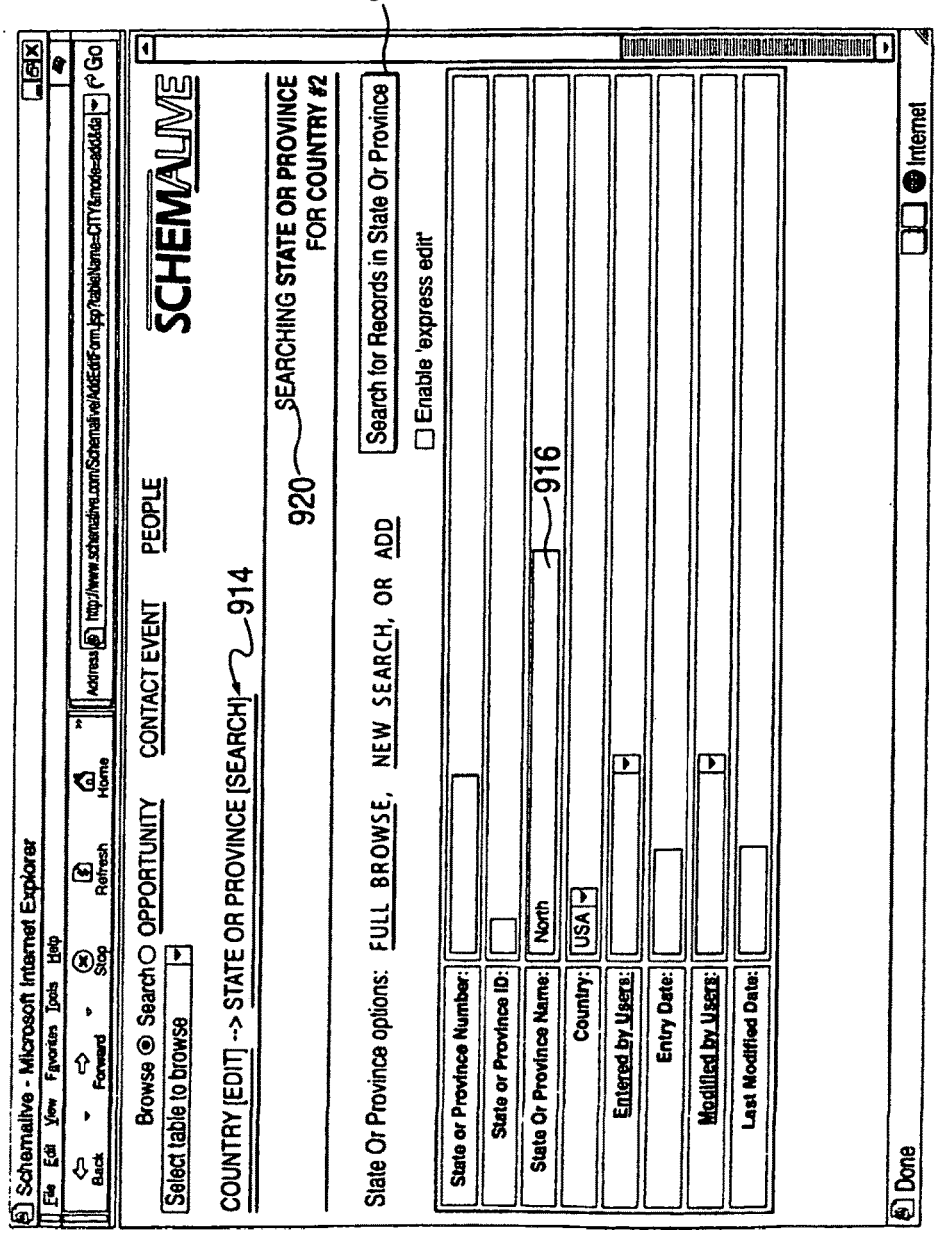

Schemalive - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://www.schemalive.com/Schemalive/MdEditForm.jsp?tableName=CITY&mode=add&da
Go
Browse Search OPPORTUNITY CONTACT EVENT PEOPLE
SCHEMALIVE
Select table to browse
COUNTRY [EDIT] --> STATE OR PROVINCE [SEARCH]
914
920
SEARCHING STATE OR PROVINCE FOR COUNTRY #2
State Or Province options: FULL BROWSE, NEW SEARCH, OR ADD
Search for Records in State Or Province
Enable 'express edit'
918
916
State or Province Number:
State or Province ID:
State Or Province Name:
North
Country:
USA
Entered by Users:
Entry Date:
Modified by Users:
Last Modified Date:
Done
Internet

FIG. 9D

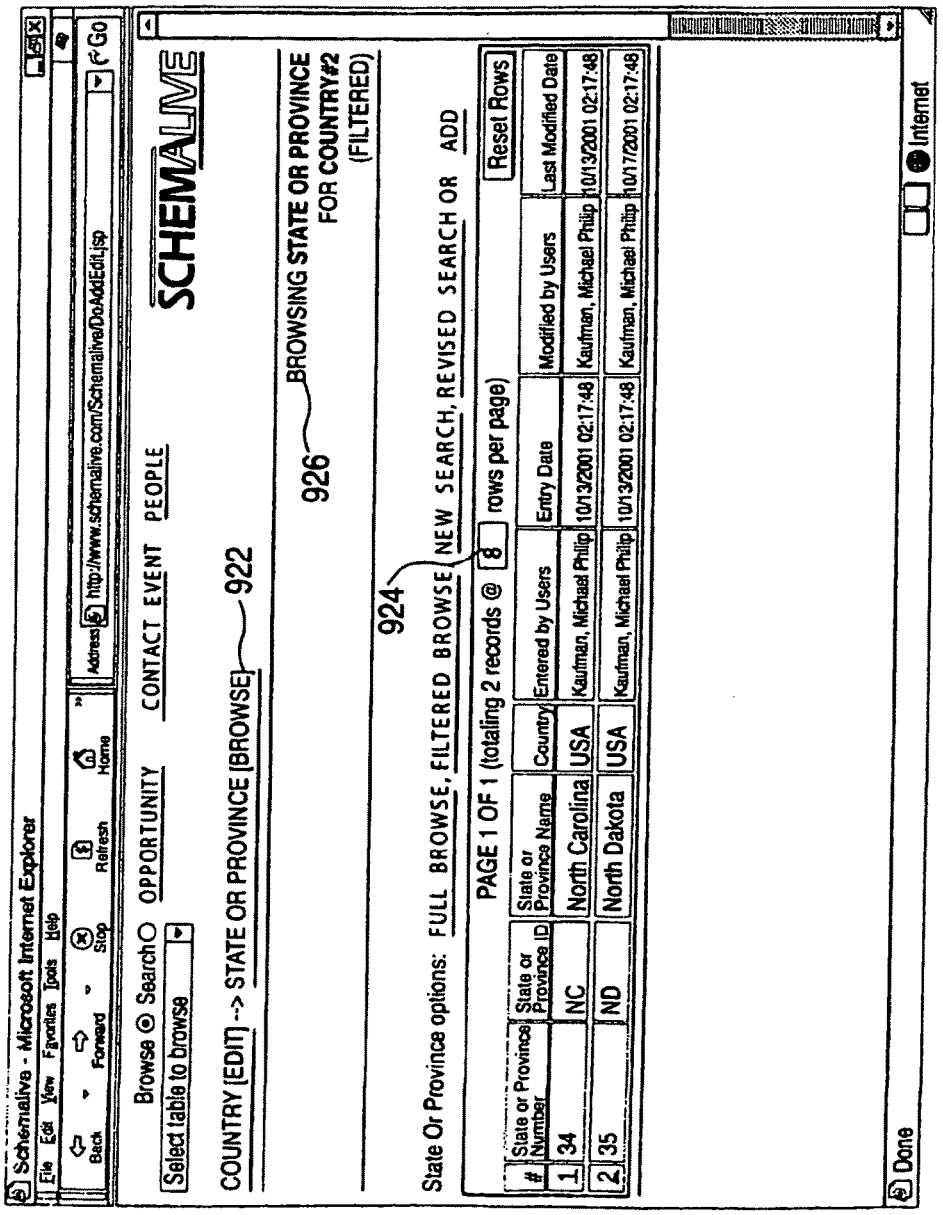

Schemalive - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://www.schemalive.com/Schemalive/DoAddEdit.jsp
Go
Browse Search OPPORTUNITY CONTACT EVENT PEOPLE
SCHEMALIVE
COUNTRY [EDIT] --> STATE OR PROVINCE [BROWSE]
922
926
BROWSING STATE OR PROVINCE FOR COUNTRY#2 (FILTERED)
924
State Or Province options: FULL BROWSE, FILTERED BROWSE NEW SEARCH, REVISED SEARCH OR ADD
PAGE 1 OF 1 (totaling 2 records @ 8 rows per page)
Reset Rows
State or Province Number
State or Province ID
State or Province Name
Country
Entered by Users
Entry Date
Modified by Users
Last Modified Date
1 34 NC North Carolina USA Kaufman, Michael Philip 10/13/2001 02:17:48 Kaufman, Michael Philip 10/13/2001 02:17:48
2 35 ND North Dakota USA Kaufman, Michael Philip 10/13/2001 02:17:48 Kaufman, Michael Philip 10/17/2001 02:17:48
Done
Internet

FIG. 9E

SYSTEMS AND METHODS FOR AUTOMATICALLY GENERATING USER INTERFACE ELEMENTS FOR COMPLEX DATABASES

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application is a continuation of U.S. patent application Ser. No. 14/324,414, filed Jul. 7, 2014, now U.S. Pat. No. 10,025,801), which is a continuation of U.S. patent application Ser. No. 13/385,913, filed Mar. 14, 2012, now U.S. Pat. No. 8,775,478, which is a continuation of U.S. patent application Ser. No. 12/930,849, filed Jan. 19, 2011, now U.S. Pat. No. 8,161,081, which is a divisional of U.S. patent application Ser. No. 11/925,236, filed Oct. 26, 2007, now U.S. Pat. No. 7,885,981, which is a continuation of U.S. patent application Ser. No. 10/428,209, filed Apr. 30, 2003, now U.S. Pat. No. 7,318,066, which is a continuation of International Patent Application No. PCT/US01/42867, filed Oct. 31, 2001, which claims priority to U.S. patent application Ser. No. 09/703,267, filed Oct. 31, 2000, now abandoned, and U.S. Provisional Patent Application Ser. No. 60/276,385, filed Mar. 16, 2001. The entire disclosure of each of the foregoing patents and patent applications, including without limitation the written description, abstract, claims, drawings, and CDROM Appendix in each such patent and patent application, and the computer source code and scripts set forth at pages 43-222 of the specification on file in the application Ser. No. 10/428,209, are hereby incorporated by reference herein.

[0002] The computer program listing appendix submitted on two compact discs in parent U.S. patent application Ser. No. 14/324,414 is hereby incorporated by reference. The compact discs contain the following directory structure:

5

File Name and Path	Date of Creation	Size in Bytes
PROV 2001 SRC/AUIFACOLD/AddEditForm.jsp	Mar. 15, 2001	24,564
PROV 2001 SRC/AUIFACOLD/Browse.jsp	Mar. 16, 2001	23,324
PROV 2001 SRC/AUIFACOLD/DoAddEdit.jsp	Jul. 2, 2014	11,312
PROV 2001 SRC/AUIFACOLD/DoViewGenerator.jsp	Mar. 14, 2001	1,486
PROV 2001 SRC/AUIFACOLD/Error500.jsp	Mar. 14, 2001	3,337
PROV 2001 SRC/AUIFACOLD/ExpiredSession.jsp	Mar. 14, 2001	3,625
PROV 2001 SRC/AUIFACOLD/OutOfSequence.jsp	Mar. 14, 2001	3,810
PROV 2001 SRC/AUIFACOLD/showSession.jsp	Mar. 14, 2001	5,032
PROV 2001 SRC/AUIFACOLD/common/EmptyParamCheck.jsp	Mar. 14, 2001	564
PROV 2001 SRC/AUIFACOLD/common/EntryPoints.jsp	Mar. 15, 2001	159
PROV 2001 SRC/AUIFACOLD/common/GlobalFooter.jsp	Mar. 14, 2001	89
PROV 2001 SRC/AUIFACOLD/common/GlobalHeaderHTML.jsp	Mar. 14, 2001	8,230
PROV 2001 SRC/AUIFACOLD/common/GlobalHeaderVARS.jsp	Mar. 14, 2001	528
PROV 2001 SRC/AUIFACOLD/WEB-INF/classes/common/debug.java	Mar. 14, 2001	1,248
PROV 2001 SRC/AUIFACOLD/WEB-INF/classes/dbUtils/CustomCaps.java	Mar. 14, 2001	211
PROV 2001 SRC/AUIFACOLD/WEB-INF/classes/dbUtils/CustomDrillDown.java	Mar. 14, 2001	1,215
PROV 2001 SRC/AUIFACOLD/WEB-INF/classes/dbUtils/CustomDropDown.java	Mar. 14, 2001	1,227
PROV 2001 SRC/AUIFACOLD/WEB-INF/classes/dbUtils/CustomDropDownComponent.java	Mar. 14, 2001	868
PROV 2001 SRC/AUIFACOLD/WEB-INF/classes/dbUtils/DataDictionary.java	Mar. 14, 2001	3,766
PROV 2001 SRC/AUIFACOLD/WEB-INF/classes/dbUtils/DataDictionaryServlet.java	Mar. 14, 2001	5,844
PROV 2001 SRC/AUIFACOLD/WEB-INF/classes/dbUtils/DataDictionaryTD.java	Mar. 14, 2001	8,238
PROV 2001 SRC/AUIFACOLD/WEB-INF/classes/dbUtils/MasterDetail.java	Mar. 14, 2001	2,412
PROV 2001 SRC/AUIFACOLD/WEB-INF/classes/dbUtils/MasterDetailServlet.java	Mar. 14, 2001	3,547
PROV 2001 SRC/AUIFACOLD/WEB-INF/classes/dbUtils/SQLUtil.java	Mar. 14, 2001	1,690
PROV 2001 SRC/AUIFACOLD/WEB-INF/classes/dbUtils/TableDescriptor.java	Mar. 14, 2001	19,181
PROV 2001 SRC/AUIFACOLD/WEB-INF/classes/dbUtils/ViewGenerator.java	Mar. 15, 2001	16,539
PROV 2001 SRC/AUIFACOLD/WEB-INF/classes/HTMLUtils/TableDescriptorDisplay.java	Mar. 15, 2001	26,692
PROV 2001 SRC/AUIFACOLD/WEB-INF/sessionUtils/ManageSession.java	Mar. 14, 2001	1,246
PROV 2001 SRC/AUIFACOLD/WEB-INF/sessionUtils/StackElement.java	Mar. 14, 2001	4,691
PROV 2001 SRC/AUIFACOLD/WEB-INF/sessionUtils/StackTag.java	Mar. 15, 2001	7,282
PROV 2001 SRC/AUIFACOLD/WEB-INF/sessionUtils/StackTagExtrainfo.java	Mar. 14, 2001	600
PROV 2001 SRC/AUIFACOLD/WEB-INF/taglib/DisplayStack.tld	Mar. 11, 2001	453
PROV 2001 SRC/AUIFACOLD/WEB-INF/taglib/showstack.tld	Mar. 11, 2001	453
PROV 2001 SRC/AUIFACOLD/WEB-INF/taglib/stack.tld	Mar. 11, 2001	1,112
PROV 2001 SRC/AUIFACOLD/WEB-INF/taglib/view.tld	Mar. 11, 2001	839
PROV 2001 SRC/AUIFACOLD/WEB-INF/tagUtils/ViewTag.java	Mar. 14, 2001	2,173
PROV 2001 SRC/AUIFACOLD/WEB-INF/tagUtils/ViewTagExtrainfo.java	Mar. 14, 2001	689
PROV 2001 SRC/SQL/create-column comments.sql	Mar. 16, 2001	1,699
PROV 2001 SRC/SQL/create-indexes.sql	Mar. 16, 2001	18,492
PROV 2001 SRC/SQL/create-procedures.sql	Mar. 16, 2001	800
PROV 2001 SRC/SQL/create-sequences.sql	Mar. 16, 2001	2,730
PROV 2001 SRC/SQL/create-tables.sql	Mar. 16, 2001	15,742
PROV 2001 SRC/SQL/create-table comments.sql	Mar. 16, 2001	3,581
PROV 2001 SRC/SQL/create-views.sql	Mar. 16, 2001	851
PROV 2001 SRC/SQL/generate-indexes.sql	Mar. 16, 2001	370
TextFiles/c-c c.sql	Oct. 31, 2000	697
TextFiles/c-p.sql	Oct. 31, 2000	288
TextFiles/c-s.sql	Oct. 31, 2000	1,241
TextFiles/c-t.sql	Oct. 31, 2000	4,495
TextFiles/c-t c.sql	Oct. 31, 2000	558

-continued

5

File Name and Path	Date of Creation	Size in Bytes
TextFiles/create-column comments.sql	Oct. 31, 2000	697
TextFiles/create-procedures.sql	Oct. 31, 2000	288
TextFiles/create-sequences.sql	Oct. 31, 2000	1,241
TextFiles/create-tables.sql	Oct. 31, 2000	4,495
TextFiles/create-table comments.sql	Oct. 31, 2000	558
TextFiles/javadoc.tgz	Oct. 31, 2000	26,011
TextFiles/SQL1.PRN	Oct. 31, 2000	6,461
TextFiles/TEMP-PRN	Oct. 31, 2000	7,587
TextFiles/AUIFACOLD/AddEditForm.jsp	Oct. 31, 2000	26,207
TextFiles/AUIFACOLD/Browse.jsp	Oct. 31, 2000	24,916
TextFiles/AUIFACOLD/DoAddEdit.jsp	Oct. 31, 2000	17,411
TextFiles/AUIFACOLD/Error500.jsp	Oct. 31, 2000	2,525
TextFiles/AUIFACOLD/showSession.jsp	Oct. 31, 2000	4,893
TextFiles/AUIFACOLD/common/GlobalFooter.jsp	Sep. 4, 2000	1
TextFiles/AUIFACOLD/common/GlobalHeaderHTML.jsp	Oct. 30, 2000	5,762
TextFiles/AUIFACOLD/common/GlobalHeaderVARS.jsp	Oct. 31, 2000	375
TextFiles/AUIFACOLD/WEB-INF/web.xml	Oct. 28, 2000	4,731
TextFiles/AUIFACOLD/WEB-INF/classes/db.properties	Oct. 31, 2000	1,221
TextFiles/AUIFACOLD/WEB-INF/classes/dbUtils/CustomDrillDown.java	Oct. 31, 2000	1,048
TextFiles/AUIFACOLD/WEB-INF/classes/dbUtils/DataDictionary.java	Oct. 31, 2000	4,216
TextFiles/AUIFACOLD/WEB-INF/classes/dbUtils/DataDictionaryServlet.java	Oct. 31, 2000	5,659
TextFiles/AUIFACOLD/WEB-INF/classes/dbUtils/DataDictionaryTD.java	Oct. 31, 2000	6,177
TextFiles/AUIFACOLD/WEB-INF/classes/dbUtils/DBConnectionManager.java	Aug. 12, 2000	14,078
TextFiles/AUIFACOLD/WEB-INF/classes/dbUtils/MasterDetail.java	Oct. 31, 2000	2,943
TextFiles/AUIFACOLD/WEB-INF/classes/dbUtils/MasterDetailServlet.java	Oct. 31, 2000	3,645
TextFiles/AUIFACOLD/WEB-INF/classes/dbUtils/SQLUtil.java	Oct. 12, 2000	757
TextFiles/AUIFACOLD/WEB-INF/classes/dbUtils/TableDescriptor.java	Oct. 31, 2000	16,747
TextFiles/AUIFACOLD/WEB-INF/classes/dbUtils/ViewGenerator.java	Oct. 12, 2000	13,510
TextFiles/AUIFACOLD/WEB-INF/classes/HTMLUtils/TableDescriptorDisplay.java	Oct. 31, 2000	23,080
TextFiles/AUIFACOLD/WEB-INF/sessionUtils/StackElement.java	Oct. 28, 2000	4,622
TextFiles/AUIFACOLD/WEB-INF/sessionUtils/StackTag.java	Oct. 30, 2000	7,045
TextFiles/AUIFACOLD/WEB-INF/sessionUtils/StackTagExtrainfo.java	Oct. 18, 2000	397
TextFiles/AUIFACOLD/WEB-INF/taglib/stack.tld	Oct. 30, 2000	1,071
TextFiles/AUIFACOLD/WEB-INF/taglib/view.tld	Oct. 27, 2000	808
TextFiles/AUIFACOLD/WEB-INF/tagUtils/ViewTag.java	Oct. 27, 2000	2,098
TextFiles/AUIFACOLD/WEB-INF/tagUtils/ViewTagExtrainfo.java	Oct. 27, 2000	671
TextFiles/Javadoc/doc/allclasses-frame.html	Oct. 31, 2000	2,291
TextFiles/Javadoc/doc/deprecated-list.html	Oct. 31, 2000	3,803
TextFiles/Javadoc/doc/help-doc.html	Oct. 31, 2000	7,327
TextFiles/Javadoc/doc/index-all.html	Oct. 31, 2000	45,124
TextFiles/Javadoc/doc/index.html	Oct. 31, 2000	785
TextFiles/Javadoc/doc/overview-frame.html	Oct. 31, 2000	1,228
TextFiles/Javadoc/doc/overview-summary.html	Oct. 31, 2000	4,557
TextFiles/Javadoc/doc/overview-tree.html	Oct. 31, 2000	6,950
TextFiles/Javadoc/doc/package-list	Oct. 31, 2000	44
TextFiles/Javadoc/doc/packages.html	Oct. 31, 2000	671
TextFiles/Javadoc/doc/serialized-form.html	Oct. 31, 2000	15,314
TextFiles/Javadoc/doc/stylesheet.css	Oct. 31, 2000	1,269
TextFiles/Javadoc/doc/dbUtils/CustomDrillDown.html	Oct. 31, 2000	10,618
TextFiles/Javadoc/doc/dbUtils/DataDictionary.html	Oct. 31, 2000	9,622
TextFiles/Javadoc/doc/dbUtils/DataDictionaryServlet.html	Oct. 31, 2000	12,203
TextFiles/Javadoc/doc/dbUtils/DataDictionaryTD.html	Oct. 31, 2000	12,429
TextFiles/Javadoc/doc/dbUtils/DBConnectionManager.html	Oct. 31, 2000	11,382
TextFiles/Javadoc/doc/dbUtils/MasterDetail.html	Oct. 31, 2000	8,946
TextFiles/Javadoc/doc/dbUtils/MasterDetailServlet.html	Oct. 31, 2000	12,231
TextFiles/Javadoc/doc/dbUtils/package-frame.html	Oct. 31, 2000	1,454
TextFiles/Javadoc/doc/dbUtils/package-summary.html	Oct. 31, 2000	5,828
TextFiles/Javadoc/doc/dbUtils/package-tree.html	Oct. 31, 2000	5,648
TextFiles/Javadoc/doc/dbUtils/SQLUtil.html	Oct. 31, 2000	8,560
TextFiles/Javadoc/doc/dbUtils/ableDescriptor.html	Oct. 31, 2000	32,949
TextFiles/Javadoc/doc/dbUtils/ViewGenerator.html	Oct. 31, 2000	10,364
TextFiles/Javadoc/doc/HTMLUtils/JspBase.html	Oct. 31, 2000	9,075
TextFiles/Javadoc/doc/HTMLUtils/package-frame.html	Oct. 31, 2000	832
TextFiles/Javadoc/doc/HTMLUtils/package-summary.html	Oct. 31, 2000	4,654
TextFiles/Javadoc/doc/HTMLUtils/package-tree.html	Oct. 31, 2000	4,805
TextFiles/Javadoc/doc/HTMLUtils/TableDescriptorDisplay.html	Oct. 31, 2000	20,243
TextFiles/Javadoc/doc/sessionUtils/DisplayStackTag.html	Oct. 31, 2000	11,387
TextFiles/Javadoc/doc/sessionUtils/DisplayStackTagExtrainfo.html	Oct. 31, 2000	8,994
TextFiles/Javadoc/doc/sessionUtils/ManageSession.html	Oct. 31, 2000	9,547
TextFiles/Javadoc/doc/sessionUtils/package-frame.html	Oct. 31, 2000	1,153
TextFiles/Javadoc/doc/sessionUtils/package-summary.html	Oct. 31, 2000	5,248

-continued

5

File Name and Path	Date of Creation	Size in Bytes
TextFiles/Javadoc/doc/sessionUtils/package-tree.html	Oct. 31, 2000	5,375
TextFiles/Javadoc/doc/sessionUtils/StackElement.html	Oct. 31, 2000	25,403
TextFiles/Javadoc/doc/sessionUtils/StackTag.html	Oct. 31, 2000	17,793
TextFiles/Javadoc/doc/sessionUtils/StackTagExtrainfo.html	Oct. 31, 2000	8,777
TextFiles/Javadoc/doc/tagUtils/package-frame.html	Oct. 31, 2000	817
TextFiles/Javadoc/doc/tagUtils/package-summary.html	Oct. 31, 2000	4,534
TextFiles/Javadoc/doc/tagUtils/package-tree.html	Oct. 31, 2000	4,643
TextFiles/Javadoc/doc/tagUtils/ViewTag.html	Oct. 31, 2000	10,815
TextFiles/Javadoc/doc/tagUtils/ViewTagExtrainfo.html	Oct. 31, 2000	8,738
TextFiles/source/AddEditForm.jsp	Oct. 31, 2000	25,845
TextFiles/source/AEF.jsp	Oct. 31, 2000	25,845
TextFiles/source/Browse.jsp	Oct. 31, 2000	24,916
TextFiles/source/DAE.jsp	Oct. 31, 2000	17,411
TextFiles/source/DoAddEdit.jsp	Oct. 31, 2000	17,411
TextFiles/source/Error500.jsp	Oct. 31, 2000	2,525
TextFiles/source/showSession.jsp	Oct. 31, 2000	4,893
TextFiles/source/ss.jsp	Oct. 31, 2000	4,893
TextFiles/source/TEMP-PRN	Oct. 31, 2000	38,622
TextFiles/source/tree.prn	Oct. 31, 2000	1,204
TextFiles/source/common/gf.jsp	Sep. 4, 2000	1
TextFiles/source/common/ghhtml.jsp	Oct. 30, 2000	5,762
TextFiles/source/common/ghvars.jsp	Oct. 31, 2000	375
TextFiles/source/common/GlobalFooter.jsp	Sep. 4, 2000	1
TextFiles/source/common/GlobalHeaderHTML.jsp	Oct. 30, 2000	5,762
TextFiles/source/common/GlobalHeaderVARS.jsp	Oct. 31, 2000	375
TextFiles/source/common/TEMP-PRN	Oct. 31, 2000	6,563
TextFiles/source/WEB-INF/TEMP-PRN	Oct. 31, 2000	12,595
TextFiles/source/WEB-INF/web.xml	Oct. 28, 2000	4,731
TextFiles/source/WEB-INF/classes/db.p	Oct. 31, 2000	1,221
TextFiles/source/WEB-INF/classes/db.properties	Oct. 31, 2001	1,221
TextFiles/source/WEB-INF/classes/TEMP-PRN	Oct. 31, 2001	7,434
TextFiles/source/WEB-INF/classes/dbUtils/CDD.jav	Oct. 31, 2000	1,048
TextFiles/source/WEB-INF/classes/dbUtils/CustomDrillDown.java	Oct. 31, 2000	1,048
TextFiles/source/WEB-INF/classes/dbUtils/DataDictionary.java	Oct. 31, 2000	4,216
TextFiles/source/WEB-INF/classes/dbUtils/DataDictionaryServlet.java	Oct. 31, 2000	5,659
TextFiles/source/WEB-INF/classes/dbUtils/DataDictionaryTD.java	Oct. 31, 2000	6,177
TextFiles/source/WEB-INF/classes/dbUtils/DBCMgr.jav	Aug. 12, 2000	14,078
TextFiles/source/WEB-INF/classes/dbUtils/DBConnectionManager.java	Aug. 12, 2000	14,078
TextFiles/source/WEB-INF/classes/dbUtils/DD.jav	Oct. 31, 2000	4,216
TextFiles/source/WEB-INF/classes/dbUtils/DDS.jav	Oct. 31, 2000	5,659
TextFiles/source/WEB-INF/classes/dbUtils/DDTD.jav	Oct. 31, 2000	6,177
TextFiles/source/WEB-INF/classes/dbUtils/MasterDetail.java	Oct. 31, 2000	2,943
TextFiles/source/WEB-INF/classes/dbUtils/MasterDetailServlet.java	Oct. 31, 2000	3,645
TextFiles/source/WEB-INF/classes/dbUtils/MD.jav	Oct. 31, 2000	2,943
TextFiles/source/WEB-INF/classes/dbUtils/MDS.jav	Oct. 31, 2000	3,645
TextFiles/source/WEB-INF/classes/dbUtils/SQLUtil.jav	Oct. 12, 2000	757
TextFiles/source/WEB-INF/classes/dbUtils/SQLUtil.java	Oct. 12, 2000	757
TextFiles/source/WEB-INF/classes/dbUtils/TableDescriptor.java	Oct. 31, 2000	16,747
TextFiles/source/WEB-INF/classes/dbUtils/TD.jav	Oct. 31, 2000	16,747
TextFiles/source/WEB-INF/classes/dbUtils/TEMP-PRN	Oct. 31, 2000	22,154
TextFiles/source/WEB-INF/classes/dbUtils/vg.jav	Oct. 12, 2000	13,510
TextFiles/source/WEB-INF/classes/dbUtils/ViewGenerator.java	Oct. 12, 2000	13,510
TextFiles/source/WEB-INF/classes/HTMLUtils/TableDescriptorDisplay.java	Oct. 31, 2000	23,080
TextFiles/source/WEB-INF/classes/HTMLUtils/TDS.jav	Oct. 31, 2000	23,080
TextFiles/source/WEB-INF/classes/HTMLUtils/TEMP-PRN	Oct. 31, 2000	39,699
TextFiles/source/WEB-INF/classes/sessionUtils/SE.jav	Oct. 28, 2000	4,622
TextFiles/source/WEB-INF/classes/sessionUtils/StackElement.java	Oct. 28, 2000	4,622
TextFiles/source/WEB-INF/classes/sessionUtils/StackTag.jav	Oct. 30, 2000	7,045
TextFiles/source/WEB-INF/classes/sessionUtils/StackTag.java	Oct. 30, 2000	7,045
TextFiles/source/WEB-INF/classes/sessionUtils/StackTagExtrainfo.java	Oct. 18, 2000	397
TextFiles/source/WEB-INF/classes/sessionUtils/STEI.jav	Oct. 18, 2000	397
TextFiles/source/WEB-INF/classes/sessionUtils/TEMP-PRN	Oct. 31, 2000	6,603
TextFiles/source/WEB-INF/taglib/stack.tld	Oct. 30, 2000	1,071
TextFiles/source/WEB-INF/taglib/TEMP-PRN	Oct. 31, 2000	7,011
TextFiles/source/WEB-INF/taglib/view.tld	Oct. 27, 2000	808
TextFiles/source/WEB-INF/tagUtils/TEMP-PRN	Oct. 31, 2000	6,918
TextFiles/source/WEB-INF/tagUtils/ViewTag.jav	Oct. 27, 2000	2,098
TextFiles/source/WEB-INF/tagUtils/ViewTag.java	Oct. 27, 2000	2,098
TextFiles/source/WEB-INF/tagUtils/ViewTagExtrainfo.java	Oct. 27, 2000	671
TextFiles/source/WEB-INF/tagUtils/VTEI.jav	Oct. 27, 2000	671

BACKGROUND OF THE DISCLOSURE

Field of the Disclosure

[0003] The present disclosure relates to the field of data processing, and more particularly to relational computer databases, and to systems and methods for automatically generating without any custom programming a user interface for the database, and/or a complete application utilizing the database, or elements thereof.

Description of the Related Art

[0004] Modern databases—and in particular, complex or large databases which serve many concurrent users—are constructed as "client/server" or "n-tier" (client/server/server) systems, wherein specialized components perform separate (and carefully delineated) functions. At a minimum, such systems are generally composed of a "back-end" relational database management system (RDBMS)—which maintains and manipulates information according to requests submitted by other components or software processes (or expert human administrators) via open-standard query languages (i.e., SQL)—and a "front-end" presentation layer or user interface, which mediates the end-users' work with the back-end data.

[0005] Developing such a database system consists both in defining the organizational structure to be used by the back-end for storing data (that is, the complement of tables which store data, and the relational links between these tables)—known as a "schema" or "data model"—and in building a front-end program (or "application") via which end-users can manipulate this data (and which communicates with the back-end on the users' behalf). And although the back- and front-end components must be closely synchronized and reflect similar structures, these respective development efforts are typically rather separate—with the requisite synchronization and parallels in structuring being effected only manually.

[0006] Moreover, the construction of front-end applications is generally undertaken using conventional third- or fourth-generation computer languages, which require by-hand coding at a very low level of functionality. Current tools for easing the development burden are limited to fairly specific (and, still, fairly low-level) uses—among them, providing more-sophisticated or "richer" controls for manipulating individual data elements; associating individual user-interface elements with specific back-end storage locations; or—at best—offering "form generator" or "wizard" facilities to automatically generate the code for a simple UI display which manipulates a single underlying (back-end) data table.

[0007] Even with such tools, considerable work remains in building a complete, fully-functional UI for a back-end schema of any appreciable size or complexity—especially where industrial-grade performance and reliability is required. And as enterprise-scale data models continue to grow, the attendant explosion of manual-coding requirements quickly becomes unwieldy—and eventually, untenable.

BRIEF SUMMARY OF THE DISCLOSURE

[0008] One object of the present disclosure is to provide a complete and fully functional user interface (UI) for any arbitrarily complex or large database schema, without any custom software programming.

[0009] A second aspect of the disclosure is that, once a back-end schema has been designed and constructed within the RDBMS, such a system can automatically "interrogate" this schema, and "absorb" its structure into an internal cache (or, at the cost of real-time performance, the internal caching mechanism can be sidestepped).

[0010] Another aspect of the disclosure is to provide a system that presents to end-users, for any arbitrarily complex or large database, a comprehensive application through which the back-end can be operated, and through which all conventional database activities—searching, listing, adding, editing—can be supported, across all base-tables comprising the schema.

[0011] In another aspect of the disclosure, an application so presented reveals (and enforces) the relational/hierarchical organization among the tables within the back-end via smoothly integrated UI mechanisms which are embedded directly into the base-table screen displays—providing a natural, powerful, and easy-to-use environment for managing complex data relationships and interactions.

[0012] One embodiment (the "reference implementation") described herein as an example of a system which may be implemented in accordance with the techniques and principles described in this disclosure, provides a system, currently written in Java and JSP, which automatically and dynamically ("on-the-fly") generates (in HTML, Javascript, and HTTP/CGI code), a fully functional UI system, based upon, and connected directly to, the underlying data model (as instantiated within an Oracle8i SQL RDBMS). The UI in this embodiment is built based on an automated interrogation of the RDBMS, either as needed (on-the-fly) or by building an in-memory representation of the data model. The generated UI in this embodiment comprises all mode displays (e.g., browse, search, edit, and add) for all tables, and a full complement of mechanisms, integrated into the mode displays for representing, navigating, and managing relationships across tables. This embodiment has the capability of creating such a UI where the underlying RDBMS is complex and comprises a plurality of tables, constraints, and relationships. It utilizes a hierarchical "context stack" for maintaining (and suspending) the working state of a particular table (comprising selected record, display "mode", pending form-field entries, in-effect search-filter parameters, Browse-mode scroll position, and any filter constraints imposed from above stack contexts) while "drilling down" across relationships to work with related information (in a possibly constrained working context) and returning relevant changes to the parent-context table, and a corresponding UI convention for displaying and navigating this stack. The embodiment provides a set of rules for traversing/navigating the context stack. It further provides naming conventions and annotational methods for enhancing and extending the representation of table structures, constraints, and relationships within the back-end so as to more fully support revelation of the schema structure through external interrogation.

[0013] Other aspects of the disclosure include, for example, techniques for automatically constructing a representation of any database table, wherein all cross-table relationships are resolved so as to supplant internal key fields in the primary table with corresponding descriptive fields derived from the related tables.

[0014] Further aspects and applications of the disclosed subject matter will be apparent to those skilled in the art from the drawings and detailed description that follow.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015] The following briefly describes the accompanying drawings:

[0016] FIG. **1** is a normal "browse mode" display from the reference implementation.

[0017] FIG. **2** is a normal "search mode" display from the reference implementation.

[0018] FIG. **3** is a normal "edit mode" display from the reference implementation.

[0019] FIG. **4** is a normal "add mode" display from the reference implementation.

[0020] FIGS. **5**A-**5**W are diagrams of the demonstration RDBMS schema from the reference implementation.

[0021] FIG. **6** is a diagram of the relationship types comprised in the paradigm of the present disclosure.

[0022] FIG. **7** is an annotated screen dump showing the active elements in a "browse mode" display.

[0023] FIG. **8** is an annotated screen dump showing the active elements in an "edit" "add" or "search" mode display.

[0024] FIGS. **9**A-**9**E show an exemplary "master/detail drill-down" and a doubly-constrained subordinate table search as rendered in the reference implementation.

[0025] In addition, the complete source code for the reference implementation, and scripts for creating the reference demonstration schema (and demonstrating the extended back-end annotational methods employed) are set forth in the computer program listing appendix (which has been incorporated herein by reference as stated above).

DETAILED DESCRIPTION

[0026] The detailed description set forth below is intended to describe various exemplary configurations of the subject technology and is not intended to represent the only configurations in which the subject technology may be practiced. The appended drawings are incorporated herein and constitute a part of the detailed description. The description includes specific details for the purpose of providing a thorough understanding of the subject technology. Numbered lists, as used throughout the description, are meant only to convey grouping and hierarchy among related items, and should not be construed to imply any sequential or ordinal significance. Applicant believes that the features and functional characteristics of the subject technology reflect novel and nonobvious advances over the prior art, and that implementations to achieve the specified features that function in the manner described herein are not limited to the specific details set forth herein.

[0027] One embodiment of the disclosed subject matter (the "exemplary embodiment"), as illustrated in FIGS. **1** through **9**E, corresponds in most respects to an implementation of this subject matter being developed under the trademark SCHEMALIVE™ which is herein referred to as the "reference implementation." A working and substantially refined embodiment of this subject matter, in the versions in which it existed on the effective filing dates of this disclosure, is further represented substantially in full by the reference-implementation source code files, documentation and scripts in the appendices accompanying and incorporated by reference into this application, as further described in the text that follows. Furthermore, the exemplary embodiment as disclosed herein also includes, in addition, some further developments that have not as yet been rendered in the reference implementation as of said effective filing dates, but which are described herein in detail and thereby constructively reduced to practice.

[0028] As can be more fully appreciated by studying the accompanying source code, the exemplary embodiment operates in accordance with a comprehensive and formalized paradigm for presenting a(n end-)user interface to any arbitrarily large or complex relational database schema (or "data model"), as represented via generally accepted data-modeling conventions (comprising the explicit declaration of any cross-table "referential integrity" [RI] constraints, and full exploitation of available native-RDBMS datatype- and constraint-attribute declaration mechanisms) and instantiated within a commercial-grade SQL RDBMS engine (Oracle8i, for example, in the reference implementation). The paradigm encompasses:

[0029] 1) A set of "modes" for interacting with a(ny) given database table (which modes, taken together, cover all desired end-user operations which may be effected upon such tables), and a corresponding display format ("screen" or "window" architecture) for each mode. These modes comprise:

[0030] 1.1) BROWSE (full or filtered, possibly context-constrained) (see FIG. **1**)

[0031] 1.2) SEARCH (new or revised, full or context-constrained) (see FIG. **2**)

[0032] 1.3) EDIT (full or context-constrained) (see FIG. **3**)

[0033] 1.4) ADD (full or context-constrained) (see FIG. **4**) Certain key screen elements for navigation control/support are shared across all of these displays (see FIGS. **7**-**8**):

[0034] 1.5) A TITLE BAR **712**, **814**, which indicates current mode, current table, context-constraint (if any), and filter indicator (if search-filter is in effect)

[0035] 1.6) A TABLE-NAVIGATION HEADER **702**, **802**, which provides direct "random access" to any system table, in either Browse or Search mode, via either a full (dropdown)—list of all (available) system tables or a short list of (clickable) "quick links" to key tables. Use of this header will also reset (and abandon) any nested stack-contexts in effect

[0036] 1.7) A CONTEXT-STACK DISPLAY **704**, **804**, which indicates the active table and mode at each level in the context stack (described below), and also allows direct navigation ("pop-up") to any suspended ("higher") stack-level (with abandonment of all lower levels)

[0037] 1.8) A MODE-NAVIGATION BAR **710**, **812**, which allows the user to move amongst the various available mode displays for the current working table (or "stack level"). The list of available modes varies, dynamically, according to both the user's access rights (described below) and the current state of the working session (i.e., whether a search-filter is currently in effect). The full list of possible mode-navigation options is: FULL BROWSE, FILTERED BROWSE, NEW SEARCH, REVISED SEARCH, and ADD. Note that FILTERED BROWSE and REVISED SEARCH appear only when a search-filter is currently in effect; if so, the former restores

a Browse-mode display with the most recent filter and scroll-position, while the latter pre-populates a Search-mode display with the current filter parameters

[0038] 1.9) Additional MODE-NAVIGATION **706** to allow "edit mode" for a single table record SCROLL NAVIGATION **708** allowing a(n end) user to navigate through all the records in a table and also allowing the user to dynamically change the number of records contained in the webpage displayed (i.e., dynamic page-sizing) HOT LINK **806** for "drill-down" to cross-reference table (e.g., in the embodiment shown in FIG. **8**, "Country") HOT LINK **808** for "drill-down" to master-detail table (e.g., in the embodiment shown in FIG. **8**, "City") CROSS-REFERENCE FIELD **810** to generate dropdown lists of available foreign-key values (with automatic correlation to display-name labels) FIELD **811** for free-form text entry, to provide automatic client-side data validation according to back-end datatype (for edit/add mode only) SUBMIT BUTTON **816** commits changes, and executes appropriate mode-switch (and stack-context return, if appropriate)

[0039] Note that, although not shown in the reference implementation, DELETE capability is also readily incorporated—as either (or both) true record-removal from the underlying table, and/or record "flagging" for UI suppression (with continued underlying-table record retention)—simply by adding (according to the user's access rights, potentially) another pushbutton within the Edit-mode display

[0040] 2) A set of rules and methods for moving among the modes (and, hence, displays) for a given table (see "mode navigation" in FIG. **7**), comprising:

[0041] 2.1) Explicit (manual) mode-selection via the mode-navigation bar

[0042] 2.2) Browse-to-Edit mode-transition for a specific record, by clicking on a Browse-row's left-most-column "row label" link

[0043] 2.3) Implicit return-to-Browse transitions from other modes:

[0044] 2.3.1) From Edit mode, upon record commit (UPDATE pushbutton)

[0045] 2.3.2) From Add-mode, upon record commit (ADD pushbutton), with optional override via an on-screen checkbox setting which "locks" user into Add mode for the current table until checkbox is cleared, or until user explicitly navigates away

[0046] 2.3.3) From Search mode, upon filter commit (SEARCH pushbutton), with optional override via an on-screen checkbox setting which enables direct Search-to-Edit transitions for single-row result-sets, provided user has requisite edit rights. In the reference implementation, this checkbox setting is session-persistent (that is, it remains in effect until the user's session terminates, so long as the user does not explicitly turn it off); it could as easily be made "sticky" to a variety of degrees—lasting for only a single search, for a single stack-context session, or even across system sessions (via database-stored user "preferences")

[0047] 3) A set of "relationship types" between individual database tables (which types, taken together, cover all desired connections between any two tables), and a corresponding UI convention for representing each type of relationship "in-place" within the (single-table) mode displays. As shown in FIG. **6**, these "relationship types" comprise:

[0048] 3.1) CROSS-REFERENCE **602** (a.k.a. "foreign key" or "FK")—single primary-table record keeps pointer to any single foreign-table record

[0049] 3.2) MASTER/DETAIL **604** (a.k.a. "parent/child" or "one-to-many")—multiple foreign-table records keep pointers to single primary-table record

[0050] 4) A set of rules and methods both for extending the representation of any single table (according to its relationships to other tables) (FIGS. **7** and **8**), and for managing (and navigating across) these relationships (comprising the resolution, display, and manipulation of cross-referenced elements within a primary table's display context, and the creation or revision of related-table information within the context of a primary table by "drilling down" to a secondary table, constraining the "working context" of that secondary table as necessary, and "passing back" relevant changes to the primary-table context) (see FIG. **9**). Said rules and methods comprise:

[0051] 4.1) Foreign-key fields occurring within a table—that is, fields which contain "keys" that uniquely identify cross-referenced records from secondary (a.k.a. "foreign", or "referenced") tables—are automatically "resolved" for display purposes, so as to substitute a corresponding (and, presumably, more meaningful) "name" field from the foreign-table record (in lieu of the key value itself—which, per generally accepted data-modeling conventions, is generally intentionally devoid of intrinsic meaning):

[0052] 4.1.1) The paradigm specifies a "default" behavior for determining this name field within the foreign-table record, based (optionally) upon a combination of field-naming conventions, field datatype (i.e., character data), field constraints (i.e., unique values), and/or order of appearance within the table definition (i.e., first non-primary-key field meeting other requirements)

[0053] 4.1.2) In the reference implementation, this field is the first one whose name ends with "NAME"—or, in special-case handling for tables containing "LAST NAME", "FIRST_NAME", and "MIDDLE_NAME" columns, a composite "Last, First Middle" value. Additional special-case processing supports successive cross-referencing through multiple tables until a "NAME" field is discovered, if (and only if) intervening tables include unique-value constrained FK columns. If no name field can be resolved, the UI displays the actual key values (that is, the primary-key values from the foreign table) themselves

[0054] 4.1.3) Alternatively, the rules for determining the name field can themselves be made "soft"—that is, specified once (globally) by a system administrator, and used thereafter to drive all (default) name-field constructions. (See the discussion of naming conventions and annotational methods, below.)

[0055] 4.1.4) The default behavior for name-field resolution can also be overridden with (either or both) "global" and/or "local" custom-name definitions for specific tables, as described below (within the discussion of extensions to, and customization of, the baseline UI paradigm)

[0056] 4.1.5) Auto-resolution of display-names applies to both Browse-mode cells (where a single display-name is derived and substituted for a given foreign-key value), and Add/Edit/Search form-fields (where a dropdown list includes the display-names for all foreign-table records, and UI actions on this list are correlated to the underlying keys)

[0057] 4.2) For "master" tables in any master/detail relationships (as specified via the core complement of naming conventions and annotational methods, discussed below), record displays incorporate a "pseudo-field" for each associated detail-table, which indicates the number (i.e., count) of corresponding detail (or "child") records belonging to the displayed master (or "parent") record:

[0058] 4.2.1) In the reference implementation, the master/detail pseudo-fields are included only for Edit-mode displays (so as to allow for streamlined system logic and, therefore, improved run-time performance)

[0059] 4.2.2) Alternatively, these pseudo-fields can also be (and have been, in alternate implementations) readily incorporated into the Browse-, Search-, and Add-mode displays, at the cost of added complexity in supporting views (i.e., correlated-subqueries for Browse-mode displays) and state-management logic (i.e., transitioning to Edit mode for not-yet-completed Add-mode transactions before allowing navigation to associated detail-table contexts where the user might add dependent "child" records), and the attendant performance implications

[0060] 4.3) To enhance the run-time performance of Browse-mode displays, the system automatically generates a corresponding back-end "view" for every table, which:

[0061] 4.3.1) Resolves all FK displays, per above

[0062] 4.3.2) Incorporates any and all default-behavior overrides

[0063] 4.3.3) By rendering (and, subsequently, executing) this view in the native language of the underlying RDBMS (i.e., SQL), effectively "projects" this extended representation of the table (according to its relationships to other tables) from the software (where it is derived) back into the RDBMS environment itself, for significantly improved rendering performance and reduced network- and application-server loading

[0064] See the discussion, below, of rules and methods for traversing/navigating the context stack, for more information on the creation and revision of related-table information within the context of a primary table

[0065] 5) A set of user-interface conventions for signaling other (non-referential) data constraints, and for enforcing adherence to same, across all Add/Edit/Search forms, comprising:

[0066] 5.1) For "required" fields (i.e., underlying table-columns with "NOT NULL" CHECK constraints, in the reference implementation), the corresponding data-field labels (descriptive names appearing to the left of the entry areas) are displayed in boldface (see FIG. **3**)

[0067] 5.2) The physical width of text-entry (vs. dropdown) fields—as well as the maximum permitted length for entered text—is driven directly by the specified data-length of the underlying table columns.

[0068] 5.3) For text fields whose length-limit exceeds a certain threshold (globally defined, in the reference implementation, though potentially user-preference configurable), the on-screen field is presented as a multiline, vertically scrollable control with multiple-row visibility, rather than the default single-row (and non-scrollable) entry field. (In the reference implementation, this is an HTML "TEXTAREA" rather than an "INPUT" field.) Note that this functionality is also applied to Browse-mode table cells, so that occasional lengthy cell-entries are made scrollable (and therefore don't distort an otherwise reasonable table-layout)

[0069] 5.4) Required fields (per above)—along with numeric, date, and text fields (whose length might exceed the threshold specification described above)—also generate automatic validation logic which prompts the user to correct any erroneous or problematic data-entries locally—that is, on the end-user's (or "client") computer, before any communication with the database takes place. In the reference implementation (which is web-based), this manifests as client-side Javascript routines—along with all requisite invocation logic, automatically embedded into the appropriate entry-field specifications—which are delivered along with the (system-generated) web-page. Failed validation (upon field-exit and/or at page-submission time, depending on the type of validation) puts the "focus" back into the corresponding problem-field (or the first of several), highlights ("selects") the entire field contents, and displays an informational pop-up dialog box explaining the problem. This effectively "projects" constraint-awareness from the back-end RDBMS (where the constraints are defined) into the front-end client, for significantly improved performance and reduced network- and database-loading

[0070] 6) A hierarchical "context stack" for maintaining (and suspending) the working state of a particular table (comprising selected record, display mode, pending form-field entries, in-effect search-filter parameters, Browse-mode scroll position, and any filter constraints imposed from above stack contexts) while "drilling down" across relationships to work with related information (in a possibly constrained working context) and returning relevant changes to the parent-context table, and a corresponding UI convention for displaying and navigating this stack

[0071] 7) A set of rules and methods for traversing/navigating the context stack, among them:

[0072] 7.1) The user is always working at the "bottom" (or rightmost, within the stack display) level of the context stack. Typically (i.e., at initial system

entry, or following direct access via the table-navigation header), there is only one level in the stack (that is, no nested or suspended stack contexts are in effect)

[0073] 7.2) Changing modes for a given table (or "stack context") is referred to as "lateral" or "horizontal" movement (see, e.g., FIG. **7**) e.g., in the embodiment shown in FIG. **9**A, a click on a mode transition button **902** (shown in this example as "19") allows for a "lateral" or "horizontal" mode transition to "edit" (shown in FIG. **9**B)

[0074] 7.3) Traversing relationships (either cross-reference or master/detail) is referred to as "drill-down" (and, upon return, "pop-up") or "vertical" movement across tables (and nested stack contexts) (see, e.g., FIG. **9**) e.g., in the embodiment shown in FIG. **9**B, a click on a "drill-down" button **904** (shown in this example as "State or Province") allows for a "drill-down" to related detail records (shown in FIG. **9**C)

[0075] 7.4) Vertical navigation therefore always increases or decreases the "stack depth", while horizontal navigation merely alters the "view" of the current table—affecting only the current (bottom-most) stack level

[0076] 7.5) Drill-downs are supported by enabling "hot-linked" (or "clickable") labels for the related data fields in the primary table (stack context) (see FIGS. **9**B and C)

[0077] 7.6) A drill-down traversal "suspends" the above stack context

[0078] 7.7) Drilling-down across a cross-reference relationship imposes no "context constraints" on the lower stack context, while drilling-down across a master/detail link constrains the subordinate table to only those records "belonging" to the above stack-context table-record (see, e.g., FIG. **9**C), such that:

[0079] 7.7.1) A superseding filter is applied to all detail-table mode displays, separate from (and invisible to) any lower-context search-filters which may subsequently be applied by the user

[0080] 7.7.2) Even a "full browse" request (with no explicit search-filter) therefore shows only related child-records

[0081] 7.7.3) The title bar **912**, **920**, **926** (across all modes) separately indicates the subordinate-table context constraint with a "FOR <PARENT-TABLE> <PARENT RECORD>"-style suffix (vs. the "(FILTERED)" suffix, which indicates a user-applied search-filter)). (For example, Title Bar **912** of FIG. **9**C shows constraint from above stack context, Title Bar **920** of FIG. **9**D still shows the context-constraint, and Title Bar **926** of FIG. **9**E reflects both the above context-constraint and the presence of a current-context "filter.")

[0082] 7.7.4) In Edit mode (for a specific child-table record), the user is prevented by filtering the dropdown-list for the corresponding FK field so that it contains only the parent-record value

[0083] 7.8) Full lateral movement (mode-switching) is supported within the subordinate stack context

[0084] 7.9) User can "return" (ascend the context stack) either by "committing" a lower-level action (a database edit or addition), or by abandoning the subordinate stack context (via the context-stack display or table-navigation header). In the former case, committed changes are automatically propagated to the above stack context and displayed in the corresponding mode display (i.e., "results" are "returned") unless the user has enabled POWER ADD in the lower context; in the latter case, any pending changes are abandoned, and the above stack context is restored exactly as originally suspended

[0085] 7.10) Cross-reference drill-downs are "context sensitive" to the parent-field status: A drill-down from a blank parent-field enters the subordinate stack context in "Add" mode, while a drill-down from a non-blank parent-field enters the subordinate stack context in "Edit" mode for the already-selected (cross-referenced) secondary-table record. Nevertheless, the default drill-down mode can be "overridden" (that is, abandoned) via a lateral or horizontal mode-switch in the lower stack context. In any event (and regardless of intervening actions), a "committed" return from a subordinate stack context will always properly update the parent record

[0086] 7.11) Master/detail drill-downs generally enter the subordinate stack context in "Browse" mode, although this behavior can be modified as a "business rule" via the described customization mechanisms (see FIG. **9** and the CreateSchema.sql script)

[0087] 7.12) The user may always return directly to any suspended ("higher") stack-context by clicking on the corresponding stack-display entry **908**. Doing so effectively "pops" the stack, and abandons any work-in-progress in all lower contexts. (For the embodiment shown in FIG. **9**C, for example, clicking on "COUNTRY [EDIT]" abandons the current stack content and restores the above context exactly as originally suspended, i.e., as shown in FIG. **9**B.)

[0088] 7.13) The user may further search or filter records at the subordinate stack context level by clicking on the "New Search" link in Mode Navigation **910**. In the embodiment shown, the further search page (see, e.g., FIG. **9**D) comprises the following screen elements:

[0089] 7.13.1) STACK DISPLAY **914** which still shows the nested contexts

[0090] 7.13.2) SEARCH FIELD **916** In the embodiment shown in FIG. **9**D, search field **916** is free-form text entry, wherein the text "North" adds an additional "filter," requiring that "State or Province Name" begins with "NORTH".

[0091] 7.13.3) TITLE BAR **920** which still shows the context constraint

[0092] 7.13.4) SEARCH INITIATING BUTTON **918** which, when clicked, initiates a "lateral" or "horizontal" mode transition to (filtered) "browse" mode (see, e.g., FIG. **9**E). The embodiment shown in FIG. **9**E comprises the following screen elements:

[0093] 7.13.5) stack display **922** which still shows nested contexts

[0094] 7.13.6) TITLE BAR **926** which now reflects both the above context-restraint (as shown, e.g., in FIG. **9**D) and the presence of current-context "filter"

[0095] 7.13.7) SCROLL NAVIGATION **924** allowing the user to navigate through all the records in a table and also allowing the user to dynamically change the number of records displayed. In the embodiment shown in FIG. **9**E, manipulating the Scroll Navigation **924** has no effect because all the records under the current constraint and filter are displayed on one page, since only two rows now meet both parent-context constraint and the current "filter."

[0096] 8) Integrated, group-based security mediation, "granular" both in scope (i.e., global-, table-, row-, or field-level) and by task (browse, edit, add, delete), which dynamically adjusts all system displays (throughout the entire UI paradigm) according to the user's granted access-rights, such that prohibited options are always hidden

[0097] Note, finally, that while the exemplary embodiment operates according to the particular paradigm described above, it remains possible to effect alternate paradigms which would nevertheless be consistent with the basic principles of this disclosure. For instance, it may be desirable in some instances to realize instead a "modeless" UI paradigm, such that all end-user activities (browsing, searching, editing, adding) are supported by a single, unified display context (such as a "spreadsheet" display).

[0098] Software (written in Java and JSP, in the reference implementation) automatically and dynamically ("on-the-fly") generates a fully functional UI system (written in HTML, Javascript, and HTTP/CGI in the reference implementation) based upon, and connected directly to, the underlying data model (as instantiated within the RDBMS), and in full conformance to the described paradigm. In order to generate the UI, the RDBMS is first interrogated or scanned by this software, applying a body of rules to interpret the data model (comprising its tables; their column-complements, datatypes, and constraints; and relationships across the tables), and to correlate same to the UI paradigm (either "on-the-fly", or by building an in-memory representation, or "cache", of said data model, and by automatically deriving enhanced back-end "views" of all tables, which are consistent with the paradigm and which, further, coherently incorporate any and all extensions, customizations, adaptations, or overrides which may have been specified as described below). In the reference implementation, the results of this RDBMS interrogation are used to construct an internal object representation of the schema, conforming to a graph in which the nodes represent database tables, and the edges represent relationships (i.e., referential integrity links) between these tables. As the UI is rendered for any given database table, this underlying object representation is referenced, and appropriate components for depicting and traversing all cross table links are automatically included in the resulting display.

[0099] A core complement of naming conventions and annotational methods (written in XML, in the reference implementation) is used for enhancing and extending the representation of the table structures and relationships (entirely within the back-end representation of the data model, in the reference implementation) so as to more fully support revelation of the schema structure through external interrogation. Said methods consist of "annotations" (or "comments") which are "attached to" (or "associated with") individual tables or table-columns within the back-end RDBMS; in discussing these methods, it is important to note that although there are any number of alternative embodiments for the formatting, storage, and association of such annotations with their corresponding objects—including (but not limited to): formatting as XML-tagged, name/value-paired, or fixed-sequence data; storage within native-RDBMS "comment" fields, application-defined database tables, or external (operating system) disk files; and association via native-RDBMS comment "attachment", explicit object-naming (within the annotations themselves), or pointers or keys (attached to the objects themselves)—the methods ultimately concern the principles by which such embodiments may be designed and applied to illuminating the schema, rather than any particular configuration or embodiment itself. Within the reference implementation, then, the attachment of annotations, as XML-formatted "comments", directly to database objects, should be considered illustrative of, rather than essential to, the methods so described. The core conventions and methods comprise:

[0100] 1) The indication of column-datatypes not natively (or explicitly) supported by the underlying RDBMS (for example, "binary" or "yes/no" fields in the Oracle8i-based reference implementation) yet subject to special handling within the UI paradigm, via the use of specific object-name suffixes ("_FLAG", in this example)

[0101] 2) The specification of master/detail relationships between tables (as distinguished from a [reversed cross-reference relationship), by associating a table-level annotation with the master (or "parent") table, and indicating both the table name and the parent-referencing FK field for each detail table (see comments in the CreateSchema.sql script)

[0102] Following the paradigm, the generated UI comprises all mode displays for all tables, with integrated(-into-the-mode-displays) mechanisms for representing, navigating, and managing relationships across tables (comprising hierarchical context constraint/enforcement, and pass-through/"pop-up" return, or "propagation", of subordinate-context results). In rendering this UI, the exemplary embodiment applies logic to (re-)convert column- and table-names retrieved through RDBMS interrogation from all-uppercase text, if necessary (as it is with Oracle8i, in the reference implementation) into mixed-case, initial-caps text (where only the first letter of each word—or "token"—is capitalized), and to replace underscore characters with spaces. The case-restoration logic is designed to also consider a list of approved acronyms—or, more generally, "exceptions"—which, when encountered as tokens within object-name strings, are instead cased exactly as they appear in the list. (This could mean all-uppercase, all-lowercase, or any non-conventional mixture of cases, such as "ZIPcode".) This case-exceptions list is provided once, globally, for the entire system, and impacts all table- and column-name references throughout the UI presentation. (In the reference implementation, the list is defined as a string array within a public "CustomCaps" object; this object could in turn be initialized via a disk file, or a special database table.)

[0103] The software also constructs and utilizes the above-described hierarchical context stack for maintaining (and suspending) the working state of a particular table (comprising selected record, display mode, pending form-field entries, in-effect search-filter parameters, Browse-mode scroll position, and any filter constraints imposed from

above stack contexts) while "drilling down" across relationships to work with related information (in a possibly constrained working context) and returning relevant changes to the parent-context table, and a corresponding UI convention for displaying and navigating this stack (see, e.g., stack display **906** in FIG. **9**C, which displays the nested contexts). Note further that, in addition to its core function in supporting nested working contexts (and by virtue of its always being on-screen), the context stack also enables certain ancillary capabilities:

[0104] 1) Since the current context (or "table-session") always corresponds to the "bottom" of the stack (i.e., the rightmost link in the display), the user can "refresh" his current table-session by clicking on this link. This can be useful, for instance, when the user wishes to "undo" or revert numerous changes made to a current Edit- or Add-mode form (but not yet committed) without having to re-navigate to the current table and record

[0105] 2) When a system exception (security violation, internal error, etc.) occurs, the resulting error screen also incorporates a stack display. Although the default error-screen behavior is to restart the user's session after a timed delay (and thereby abandon all work in progress), the user will often be able to recover his session by making a selection from the error-page stack display

[0106] The exemplary embodiment further provides a structured collection of methods, mechanisms, tools, techniques, and facilities for extending, customizing, adapting, or overriding the baseline UI paradigm and software to support non-standard and/or special requirements ("business rules"), comprising:

[0107] 1) The capability to "override" the default behavior for FK "display-name" resolution with (either or both) "global" and/or "local" custom specifications on how to generate display-names for a given foreign-key:

[0108] 1.1) Such overrides can be useful, for example, when the foreign (referenced) table lacks a (resolvable) name column; when a composite (multiple-field), treated, or otherwise modified display-name is desired; when the sort-order within display lists should be modified; or when the foreign-table records depend on yet other table-records (foreign, in turn, to the FK-referenced table) for full name construction (for instance, where FKs into a "CITY" table depend in turn on FKs from CITY into a "STATE" table in order to distinguish like-named cities, such as Portland, Oreg. and Portland, Me.)

[0109] 1.2) A custom specification consists of an explicit SQL expression that generates key-value/display-name pairs for any and all foreign-table key values

[0110] 1.3) Such specifications will automatically propagate throughout the entire UI, including all relevant Browse-mode cells and Add/Edit/Search form-fields

[0111] 1.4) Global display-name specifications are associated as table-level annotations (see above) with the referenced foreign table

[0112] 1.5) Local specifications are associated instead as column-level annotations with the referencing (foreign-key) column in the base-table itself

[0113] 1.6) In this way, both "default" (global, or system-wide) and "special-case" (local, or single referencing-table only) custom display-names can be defined for the same foreign table. If a "local" specification is defined for a given FK-column, it will supersede any "global" or "default" specification also defined for the referenced (foreign) table.

[0114] 1.7) In the reference implementation, specifications are made via a special XML tag ("<sql>") which is attached to the table or column (for global or local specifications, respectively) as a "comment"

[0115] 2) Ability to alter the order and visibility of individual table-columns across all mode displays (Browse, Add, Edit, Search) vs. the actual column-complement and -ordering of the associated (underlying) table:

[0116] 2.1) This is sometimes desirable in a post-production environment, especially when the particular back-end RDBMS product in use makes it impractical or impossible to alter the actual structure of the underlying table once it has been populated with data and is participating in referential-integrity relationships with other populated tables

[0117] 2.2) A specification consists of a listing of the desired table-columns, in the desired display order (either by name or, alternatively, by ordinal position in the actual underlying table)

[0118] 2.3) If a specification is made, then any columns not explicitly included within that specification will be suppressed from the UI mode displays

[0119] 2.4) Specifications are associated as table-level annotations with the actual underlying table

[0120] 2.5) In the reference implementation, specifications are made via a special XML tag ("<columnOrder>") which contains sub-tags ("<cl>") indicating the desired columns in order and by name, and is attached to the table as a "comment"

[0121] 3) Support for composite or "custom views" of multiple-table data which mimic a single base-table. Such a derived (non-table) result-set is typically generated by a "stored query" or "SQL VIEW" within the back-end RDBMS, and nevertheless can be rendered and presented by the UI as if it were an actual single base-table (subject to certain limitations which may be imposed by the underlying RDBMS—particularly, the inability to edit or add "records" for such result-sets, rendering them effectively "read-only")

[0122] 4) Ability to manually define Search-mode "dropdown fields" (which list the range of possible values for a given column) for such custom views:

[0123] 4.1) Because, by its nature, the custom view appears to be an actual table—and therefore obscures the underlying (real) tables on which it is based—the system cannot automatically resolve the referential-integrity (RI) links that would normally serve to identify the appropriate value lists (i.e., foreign-table values)

[0124] 4.2) Moreover, the normal value-to-key translations managed by dropdown fields are inappropriate for custom views anyway, since these views actually incorporate the cross-referenced values themselves (rather than foreign keys that point to these values, as base-tables do)

[0125] 4.3) To support custom-view dropdown lists that (appear to) behave consistently with the general (actual-table) UI paradigms, then, a manual (explicit) dropdown-list specification is made for each corresponding custom-view column

[0126] 4.4) A specification identifies the foreign table which contains the dropdown-list values, and the column (either by name or, alternatively, by ordinal position within that table) which supplies the actual values

[0127] 4.5) Specifications are associated as column-level annotations with their corresponding custom-view columns

[0128] 4.6) In the reference implementation, specifications are made via a special XML tag ("<manualDropDown>") which, in turn, contains sub-tags indicating the related foreign-table name ("<foreignTableName>") and key field ("<foreignKeyField>"), and is attached to the corresponding view-column as a "comment"

[0129] 5) In-place pass-through (drill-down) from custom views to Edit-mode displays for underlying (component) base-table members:

[0130] 5.1) Because the "stored queries" or "SQL VIEWS" that underlie custom views are typically non-updateable (according to RDBMS limitations), the usual UI mechanisms for editing data cannot be used with these views. Nevertheless, it is often desirable to provide users with easy access to editing for (at least some of) the data behind the views

[0131] 5.2) To enable such editing access, a mechanism is provided to create a (series of) cross-referential link(s) from the individual cells (row-values) in a given column of a Browse-mode display, with each link forwarding the user to a secondary display—most commonly, to an Edit form for the underlying base-table containing that cell's value (although it is, in fact, possible to link-through to any arbitrary table, row, and column, and in any "mode")

[0132] 5.3) While such links usually reference the same underlying base-table (and -field) for every row in the column, special-case extension logic can reference different tables for different rows, according to "trigger" or "switching" values from another column in that same display-row

[0133] 5.4) A further variation of the mechanism (described below) modifies the behavior of the leftmost-column "row label" links, rather than the interior Browse-mode

[0134] 5.5) On-screen, the link appears as a highlighting (in the reference implementation, a "clickable link" or HTML "HREF") of the cell-value itself. (Empty cells display the value "NONE" so as to still enable drilldown navigation.) When the user selects (clicks on) the link, the display forwards (typically) to an Edit form for the corresponding record in the appropriate underlying base-table, with the proper edit-field pre-selected (i.e., given the "focus"). In effect, the system auto-navigates to the same exact base-table Edit form, selected-record, and edit-field that the user could (theoretically) navigate to himself, manually, in order to alter the underlying datum that supplies the custom view

[0135] 5.6) The working context for this drilled-down Edit form is constrained by the same mechanisms that govern master/detail drilldowns (as described above)—that is, a stack-context filter is imposed on the edit session in order to prevent the user from changing the datum that links the base-table record to the custom view (note that this also requires a separate, explicit specification of the base-table as a "detail table" to the custom view); and if/when the user "commits" the drilled-down edit session (by pressing the "Update" button), she is automatically returned to the "parent" custom view

[0136] 5.7) A specification identifies the underlying (or "target") base-table; the (initial) base-table display-mode (typically, "Edit"); the custom-view column whose corresponding row-value contains the identifying key for the target base-table record; the custom-view column (if any) whose corresponding row-value contains the "constraining" (master/detail) key; and the base-table field-name which should be selected (i.e., the field that contains the target value, and should therefore receive the "focus")

[0137] 5.8) Specifications are associated as column-level annotations with their corresponding custom-view columns

[0138] 5.9) A special-case extension of the specification can be associated as a table-level annotation with the custom view itself (rather than one of its columns). In this context, the specification will modify the behavior of the leftmost-column "row label" links (which, in normal-table Browse-mode displays, link to Edit-mode displays for the corresponding table-records). A common use for such specifications is to support master/detail-style transitions to secondary Browse-mode displays of records which "belong to" the selected custom-view record

[0139] 5.10) In the reference implementation, specifications are made via a special XML tag ("<customDrillDown>") which, in turn, contains sub-tags indicating the target base-table ("<tableName>"), display-mode ("<mode>"), identifying-FK field within the custom view ("<keyColumn>"), constraining-context or master/detail key, if any ("<parentColumn>"), and target field ("<focusField>"), and is attached to the corresponding view-column as a "comment"

[0140] The exemplary embodiment also supports the specification and enforcement of both global and granular (by table and function) access rights and activity-stamping, according to a group-based (rather than hierarchical) permissions scheme, and based on table entries which themselves can be entered and maintained via the system:

[0141] 1) In the reference implementation, six tables support these security features: PEOPLE, USERS, SECURITY_TABLE, SECURITY_GROUP, SECURITY_GROUP_USERS, and SECURITY_GROUP_TABLE:

[0142] 1.1) The PEOPLE table contains an Active_Flag field, which allows for "deactivation" of individuals without destroying existing RI links throughout the database. Every system user must appear in the PEOPLE table (among other reasons, to support full-name resolution when displaying usage-tracking

fields through the UI), and if/when a user's PEOPLE. Active_Flag is turned off, the user is immediately blocked from all further system access

[0143] 1.2) The USERS table incorporates (among others) a Login ID field, which is correlated against the system-user's operating-environment credentials. (In the reference implementation, this is the UID which has been authenticated and forwarded by the web server; alternatively, it could be the user's OS login.) When the system establishes a new user-session (upon the user's initial contact), it attempts this correlation to a valid USERS.Login_ID. If no correlation can be made, access to the system is denied; otherwise, the corresponding USERS.Users_Key value is henceforth associated with that user's session

[0144] 1.3) SECURITY_TABLE maintains a list of all security-mediated tables and custom views. (Alternatively, this list could be automatically derived from the system's data-model interrogation; the use of an explicit and hand-managed table supports the manual exclusion of "special" or "hidden" tables and/or views)

[0145] 1.4) SECURITY_GROUP supports the definition of functional security roles. In and of themselves, entries to the SECURITY_GROUP table are little more than descriptive names; their primary purpose is to serve as "connective conduits" between USERS and SECURITY_TABLEs. It is important to note (again) that SECURITY_GROUPs are non-hierarchical; that is, each group can be granted any mix of rights to any arbitrary set of tables, without respect to the rights of other groups. And USERS can be assigned to any number of SECURITY_GROUPs; When a user belongs to multiple groups, her aggregate rights comprise a superset of the rights for each of the groups to which she belongs

[0146] 1.5) SECURITY_GROUP_USERS simply effects many-to-many relationships between USERS and SECURITY_GROUPs, and is defined (via the methods described above) as a "detail" table to both of these

[0147] 1.6) Similarly, SECURITY_GROUP_TABLE supports many-to-many relationships between SECURITY_GROUPs and SECURITY_TABLEs (and is a "detail" table to both). Additionally, however, the SECURITY_GROUP_TABLE incorporates Boolean (true/false) columns which indicate permission for the related SECURITY_GROUP to (respectively) browse, add to, edit, or delete from the corresponding SECURITY_TABLE. This forms the *nexus* of access-rights control

[0148] 2) All UI displays automatically adjust to the current user's access rights. In particular, the following navigational elements ("links", as defined in the reference implementation), appear or are suppressed according to the user's rights:

[0149] 2.1) Mode-navigation bar links **710** (browses/searches/add); here, suppressed links are entirely removed from the display, rather than simply "disabled" (or made "non-clickable", as is done for all other links, below)

[0150] 2.2) Record-edit links **706** (in the first column of Browse-mode displays)

[0151] 2.3) Drill-through cross-reference links (on the labels of Add/Edit/Search dropdown fields)

[0152] 3) Drill-down master/detail links (on the labels of Edit-form master/detail summary-counts)

[0153] 4) Note that custom views with custom-drill-down specifications are subject to "double" security mediation: If edit permission to the custom view itself is withheld for a given user, then all custom-drilldown links will also be disabled. But (even) if the custom-view edit permission is granted, the user must also have the necessary rights to support each particular drill-down (e.g., edit or browse permission on an underlying table) before the corresponding link will be enabled

[0154] 5) Separately (and assuming the necessary access rights have been granted), all system add/edit activity can be time- and user-stamped at the table-record level (optionally, on a per-table basis). Security-stamping is completely automatic, and is governed (in the reference implementation) by the presence of four special columns within the table: Entered_By_Users_Key, Entry_Date, Modified_By_Users_Key, and Last_Modified_Date. If these columns exist, then any "add" event causes the current USERS.Users_Key (from the user's session) to be recorded in both the Entered_By_Users_Key and Modified_By_Users_Key columns, and the current system time to be stamped into both the Entry_Date and Last_Modified_Date columns. "Edit" events, of course, update only the Modified_By_Users_Key and Last_Modified_Date columns. Note further that when they exist in a table, these fields are visible only in Browse and Search displays; they are hidden (but automatically updated) from Add and Edit displays

[0155] 6) Although not present in the reference implementation, the granularity of this model can be readily extended with both row- and column-level access mediation:

[0156] 6.1) ROW-LEVEL SECURITY allows for the individual rows (records) of any given table to be made visible or invisible (and, therefore, accessible or inaccessible) to a given user:

[0157] 6.1.1) In a sense, row-level security can be said to affect only "content" visibility, rather than "structural" visibility (as with other security axes); a row-level security filter impacts which particular table-entries are presented, but never which classes or types of data elements

[0158] 6.1.2) A specification thus identifies the filter condition (i.e., WHERE clause) that relates one or more table-columns to (some transformation/JOIN-sequence on) the current user. (Note that such "user relations" may optionally involve attributes of the particular user, and/or those of "security groups" to which the user belongs)

[0159] 6.1.3) Specifications are associated as table-level annotations with the actual underlying table

[0160] 6.1.4) Because there are no effects upon the structure or "shape" of the data, these filters can be "encapsulated", effectively, and introduced as a (logical) "shim" layer between the raw back-end tables and the data-dictionary object model.

[0161] 6.1.5) By exploiting the identical column structure of each such "shim view" to its underlying base-table, on the one hand, and to the

"virtualized" schema view (as constructed during the interrogation phase) of that table, on the other, the rest of the system logic and infrastructure can be insulated from any awareness of (or sensitivity to) this mechanism

[0162] 6.1.6) Application of the row-level filter consists of "surgical" modifications to the defining SQL for the corresponding Browse-mode view (see above), so as to incorporate the requisite additional WHERE clause (and any additional FROM-clause tables, utilizing the same view-integration and alias-merging logic already employed within the reference implementation in generating said view)

[0163] 6.1.7) Function-oriented mediation (i.e., Browse/Edit/Add/Delete granularity) is supported via (optional) separate specifications (per table) for each function (and with a "default/override" hierarchy among these specifications—such that Browse rights obtain for editing, for instance, unless explicit Edit rights have been specified). The UI-generation logic then compares record-presence across the respective (resulting) views to resolve specific rendering and action decisions (i.e., is this record editable?)

[0164] 6.2) COLUMN-LEVEL SECURITY allows user access to be governed on a field-by-field basis:

[0165] 6.2.1) Specifications are analogous to those described in the reference implementation for table-level security (see the discussion of SECURITY_GROUP_TABLE, above), except that only "Browse" and "Edit" rights are meaningful on a per-column basis (that is, there is no way to "Add" or "Delete" only individual columns)

[0166] 6.2.2) Column-level specifications are treated as "subtractive overrides" to table-level specifications, such that table-level specifications serve as "defaults" that can be further restricted—but not expanded--by column-level specifications

[0167] 6.2.3) Application of column-level security to the Browse function consists of an additional "overlay" view which hides additional columns as necessary

[0168] 6.2.4) Edit-function mediation is processed by the UI on a per-field basis, either (or both) during rendering (where display conventions utilize read-only fields, or otherwise signal non-editability via labeling conventions [such as italicized text]) and/or processing (where attempts to change non-editable fields are rejected, with an alert notification to the user)

[0169] Also incorporated into the exemplary embodiment are both generalized and special-case exception-handling mechanisms, with integrated session-recovery support:

[0170] 1) The generalized exception-handling mechanism guarantees a controlled recovery from any unanticipated error condition. This mechanism:

[0171] 1.1) Presents as much diagnostic information as possible, within a paradigm-consistent UI display, comprising:

[0172] 1.1.1) A pass-through errortext from the underlying program-execution environment

[0173] 1.1.2) A complete "(program call-)stack dump" indicating the suspended (and nested) program-function calls in effect at error-time

[0174] 1.1.3) The entire current context-stack display 1.2) Permits user recovery either by:

[0175] 1.2.1) Controlled reinitiation of a(n entirely) new session

[0176] 1.2.2) Navigation through the context-stack display to a pre-error session context, thereby (generally) enabling the user to recover his session-in-progress (more-or-less) intact, vs. requiring a restart from scratch

[0177] 2) Special-case exception-handling mechanisms are defined separately for certain types of system errors which are common or "normal" (such as authorization failures or session timeouts). In such cases, these "customized" exception-handlers can suppress unnecessary technical detail (which can be confusing or alienating to end-users and give the misimpression of software failure), and provide additional (end-user suitable) information specific to the user's particular error context. The reference implementation can identify and separately handle the following common exceptions:

[0178] 2.1) SESSION-SEQUENCE ERRORS: In the reference implementation (which, again, is web-based), it is important that the system govern the "flow" or sequence of pages passed back and forth between the (web-)server and the client (web-browser); as a result, the system incorporates several mechanisms to track and enforce this flow (comprising back-button "defeat" logic, and incremental serialization of all URLs [such that the system always knows what serial number to "expect" along with the user's next page-submission]). If the user manages to violate this flow, either intentionally or inadvertently (perhaps by selecting a "favorite" or "bookmark", or by clicking multiple links on the same page before the server can respond), the system can detect this particular error, provide a detailed explanation of how and why it might have occurred, and (per above) allow the user to recover her session-in-progress without any loss of work

[0179] 2.2) SECURITY VIOLATIONS: Generally, the system proactively prevents the user from attempting access to any authorized system modes or functions. However, in the (web-based) reference implementation, it is not impossible for the user to navigate to a situation where he might possibly attempt an illegal transition—or to manually adjust a URL so that it attempts such unauthorized access without triggering a session-sequence error (as described above). In these cases—and in the simpler case, when a user attempts access without any system rights whatsoever—the system provides a plain-English report of exactly what access rights the user has tried to violate

[0180] 2.3) SESSION TIMEOUT: Because the system maintains a "user session" in which various context, sequence, and configuration information is tracked, and which (because it consumes system resources) can expire after a (configurable) period of disuse—and also because (in the web-based reference implementation) the dialog between client and server is "connectionless" (meaning that there can

never be any automatic detection by the server that a user has "quit" or "broken" a connection)—it is entirely possible that a user may try to continue or resume a session which appears perfectly intact from his perspective (i.e., in his web-browser) but for which the system has discarded the corresponding user-session. In this case, a full session-reinitiation is still required—but it can at least be delivered along with a meaningful explanation of what has occurred

[0181] These special-case error handlers dovetail and integrate smoothly with the generalized exception-handling facility, and share many of the same features (including, when available, the session-stack display). Within the reference implementation, these handlers are hard-coded, but they describe the basis of a subsystem which can be readily extended—abstractly and dynamically—in several ways:

[0182] 2.4) Specific exceptions—and their corresponding, customized error displays—can be defined and administered via a central list (or table), and automatically detected (and their respective displays invoked) at runtime, within the framework of a generalized facility and without the need for custom programming

[0183] 2.5) Information can be "mined" from the pass-through errortext—and, potentially, from the runtime environment as well—according to the nature of the particular error, and used (if appropriate) in the construction of dynamic error displays (via templates, for example)

[0184] 2.6) Custom follow-on actions can be associated with specific errors, so that special-case recovery procedures can be specified. (For instance, a database-detected data-entry violation might cause a return to the previous data-entry form.) "Mined" runtime-environment information can also be used here to govern the behavior of said follow-on actions

[0185] A generalized, extensible, and data-driven "pop-up help" facility is also included in the reference implementation. This facility allows for the specification of descriptive text which can be associated both with specific on-screen navigational elements, and with (any) individual schema elements (i.e., table-columns). When the user positions his mouse over a described object (or data-field) and pauses for a specified timeout interval, the system will flash a pop-up window (or "balloon") displaying the corresponding description. The system thereby becomes self-documenting with respect to both the UI paradigm itself, and the meaning of its data-fields. Within the reference implementation, the specifications are stored within back-end tables—so that they, too, may be administered via the system UI—although any of the above-described annotational methods could alternatively be used.

[0186] Except as noted, the detailed implementation of each of the foregoing capabilities is set forth in full in the accompanying source code, which represents the complete source code for a working version of the reference implementation. A full demonstration RDBMS schema upon which this system can operate has been provided, and accompanies this application and is incorporated herein by reference (see FIG. **5** and the CreateSchema.sql script).

[0187] Numerous extensions of the above-described scheme are of course possible:

[0188] 1) Most importantly, while the reference implementation is in various instances custom-coded to the data-dictionary architecture of its particular underlying RDBMS (i.e., Oracle8i), the scheme is nevertheless readily converted to a "generic" (or "RDBMS-agnostic") architecture through the introduction of a platform-neutral "middleware" layer. (The DatabaseMetaData class within the Java 2 Platform Standard Edition v1.3.1 API Specification, for instance, is easily applied toward this end.) The claimed invention, therefore, is by no means limited to a specific RDBMS product

[0189] 2) A set of mechanisms, rules, and methods may be provided through which each end-user can evolve (and manage) personalizations to the UI architecture (with persistent back-end storage and tracking by user and/or group)—including (but not limited to) preferred table-navigation hierarchies; UI "entry points" based on usage-frequency patterns; default (or most-recent) searches/filters for each back-end table; default "page size" for Browse-mode lists (adjusted for the particular user's screen resolution, for example); default sort-orders for each table; and default "Power Edit" and "Power Add" settings. Because user-tracking is already integrated (for security purposes), it is a simple matter to add the supporting tables and UI-application "hooks" to collect, store, and utilize such preference information

[0190] 3) Expanded concurrency-control options are easily incorporated into the scheme. Many database-related systems offer a range of behaviors which extend from unfettered write-back of edited table-records (offering maximum system performance, at the cost of minimal overwrite protection), through competing-update detection with approval/abandonment of data overwrites (a blend of performance and protection, at the cost of added complexity), to full edit-record locking (offering maximum protection at the cost of performance); and while the reference implementation incorporates only the first of these behaviors, the others can certainly be added—along with a system-configuration mechanism for choosing among them—in a straightforward manner

[0191] 4) A generalized journaling/auditing subsystem may also be integrated. Such a subsystem could, for instance, utilize database "triggers" to update a master table with a new tuple (comprising table-name, record-key, column-name, old-value, new-value, user-key, and timestamp) whenever any table-record is modified. Such a mechanism would (at a cost in system performance, of course) permit complete backtracking/"roll-back" to previous database states, and guarantee the ability to recover from any rogue data modifications (whether accidental or malicious) and identify the actors

[0192] 5) A further extension to journaling/auditing support is the ability to require a user to explain his justification for (only) certain data-field changes, and then either record that explanation to the system journal or audit log (along with the other tuple information), or (possibly) roll-back the transaction (if the user declines to supply an explanation). Such a facility could be implemented with additional text-entry fields integrated into the primary Edit-mode display, or alternatively, with "pop-up window" logic (which, within World Wide Web presentation, could comprise addi-

tional browser windows or DHTML "simulated" pop-ups, for instance). The specification of which data-fields should require such justification would be considered a "business rule", and could be implemented via any of the annotational methods described elsewhere in this document. Such specifications could also be assigned at various levels of global vs. local "scoping" (i.e., perhaps automatically for all date fields, or only for specifically assigned text fields)

[0193] 6) Within the current (World Wide Web-based) reference implementation, it is possible to select certain navigational links (for example, from the context-stack display or the mode-navigation bar) which will abandon the user's current screen display and, with it, any data entries or modifications which may have been made but not yet committed to the database. Although this behavior is by design, it may be desirable to add a pop-up "warning" mechanism for such cases, so as to alert the user to the imminent loss of data (and to provide for aborting said action). Such a mechanism could utilize client-side Javascript logic to:

[0194] 6.1) Set an internal flag each (and every) time any on-screen change is made

[0195] 6.2) Invoke a "cover function", each time a screen-abandoning link is clicked, which will display a confirmation dialog (pop-up window) if the "change flag" has been set (or, if the flag is not set, will simply execute the link)

[0196] 6.3) Proceed with the link action (and abandon the current screen) only if the user supplies explicit confirmation

[0197] 7) A variety of extensions can be made to the Browse-mode display paradigm, comprising:

[0198] 7.1) The ability to sort Browse-mode listings (by any combination of columns) by clicking on the corresponding column-headings. Successive clicks on the same column-heading would invert the sort-order for that column; successive clicks on different columns would effectively produce "ordered sorting" (where the most-recently clicked column is the "primary" sort, and each successively less-recently clicked column is the next "subordinate" sort)

[0199] 7.2) Support for "random-access" page navigation, wherein the table-header (which, in the reference implementation, allows direct entry only for the number of rows per page) would also allow direct entry of the desired page number. For instance, a Browse-mode display whose table-header said "PAGE 5 OF 12 (TOTALING 300 RECORDS AT 25 ROWS PER PAGE)" would thus render both the "5" and the "25" as text-entry fields, so that in addition to resizing the page length (by changing the rows-per-page entry), the user could also "zoom" to a specific page just by changing the page-number entry. This would eliminate the need to scroll, page-by-page, from either the top or bottom of the result-set

[0200] 7.3) Similarly, another form of random-access page navigation could be introduced via the addition of phonebook-style "tab" links (for instance, "A|B|C|D . . . ") such that clicking a particular link would jump to the first record in the result-set whose corresponding-column entry began with that character:

[0201] 7.3.1) Said "corresponding column" could be (initially) determined according to similar default-processing rules to those embodied in the reference implementation for FK display-name resolution (for instance, the first column whose name ends in "NAME", if any)

[0202] 7.3.2) Alternatively, the corresponding column could simply track the current (primary) sort-order column (as described above), if implemented

[0203] 7.3.3) Yet another option would be to allow explicit designation of the corresponding column via an associated dropdown-list of all table-columns

[0204] 7.3.4) However selected, any change in the corresponding column would then automatically regenerate the tab list, according to the range of actual (sorted) leading characters appearing within that column. In this way, numeric tabs would appear for a "social-security number" column, vs. alphabetic tabs for a "last name" column

[0205] 8) A variety of extensions can be made to the Search-mode display paradigm, comprising:

[0206] 8.1) In the reference implementation, field-value filters are applied by default as prefix matches (i.e., as "starts with" comparisons), with optional support for explicit relational-operator prefixing (comprising <, <=, >=, >, and exactly =). Relational options could be further extended to support ranges ("between x and y"), NULL/NOT-NULL conditions, and other arbitrarily complex trans-formations on the corresponding field-values (such as field-value substitution into a complex string-manipulation or arithmetic expression)

[0207] 8.2) The reference-implementation Search-form paradigm comprises a single set of fields (corresponding to the underlying table-columns), where any entered filter-values (for the respective columns) are logically "AND"ed together. A more general and flexible search facility could:

[0208] 8.2.1) Allow toggling between logical "AND" and "OR" combination of a search form's filter-values

[0209] 8.2.2) Allow "stacking" of multiple search-form copies, such that the fields in each individual (sub-)form comprise a parenthetical filter "phrase", which is "AND"ed or "OR"ed together (selectably, as above) with the parenthetical phrases for other sub-forms

[0210] 9) A variety of extensions can be made to the Edit-mode and Add-mode display paradigms, comprising:

[0211] 9.1) In the reference implementation, violations of any extant "unique" constraints on underlying table-columns are intercepted and reported only upon violation, and then only via the generalized exception-handling mechanism (in response to a back-end RDBMS exception "throw"). Alternatively:

[0212] 9.1.1) Special-case exception handling (as described above) could still exploit the thrown back-end exception, but provide clearer diagnostics (i.e., exactly—and only—the field-value that

has violated a "unique" constraint), and then restore the data-entry form with the problem-field contents pre-selected; or

[0213] 9.1.2) Employ separate database-interrogation logic for each "unique"-constrained field, so as to "pre-qualify" data-entries—and, thereby, allow for "in-place" duplicate-entry detection and signaling (without ever leaving the data-entry form, and without invoking formal exception-handling mechanisms)

[0214] 9.2) Similarly—but more generally—violations of any arbitrary "check" constraints (such as imposed value-ranges, or required satisfaction of algebraic expressions) are intercepted and reported only upon violation within the back-end RDMBS. Instead, such constraints could be extracted from the back-end and "projected" into the client-side UI display (for the reference implementation, via custom-generated Javascript routines). Doing so would allow the detection and signaling of constraint violations immediately upon data-entry, without (additional) contact with the back-end RDBMS (and this, in turn, would obviate the need for any display/session recovery logic)

[0215] 9.3) When adding new records, the reference-implementation Add-form logic does not "initialize" fields for which the back-end defines "default" values—that is, although the underlying table-column will (properly) be set to its default value if the corresponding Add-form field is not explicitly set, the user has no indication (prior to committing the new record) of that default value. Instead, the form could automatically pre-populate the appropriate fields with their corresponding default values (as determined through interrogation of the underlying column-constraints)

[0216] 10) In certain situations, it may be desirable during schema interrogation to "deduce" relational interdependencies between tables where no explicit referential-integrity constraints have been defined. In such cases, it is possible to further compare field-names and associated attributes across tables, so as to identify columns which (for instance) are identically named, and (only) one of which is the primary key for its respective table. Under these conditions, it could (optionally) be assumed that the other-table column is a foreign-key cross-reference to the first column. Note that, in so doing, the UI paradigm would then enforce referential integrity for this relationship, even absent the explicit back-end constraint.

[0217] 11) Additional mechanisms for further customizing or adapting the baseline UI paradigm and software to meet non-standard and/or special requirements ("business rules") are also indicated, such as:

[0218] 11.1) Specification and enforcement of correlations, interactions, or interdependencies between disparate data-elements (either within or across base-tables), comprising:

[0219] 11.1.1) "Context-sensitive dropdown controls", whose dropdown-lists are filtered (or "constrained") based on user-defined relations to superior stack-contexts (other than direct master/detail constraints, which already are included as a part of the core UI paradigm). Such controls could be specified via any of the aforementioned annotational methods. Specifications would "attach" to the subordinate-level table-column (i.e., the column whose dropdowns should be "filtered" or "sensitized"), and would consist of tuples indicating (at least) the superior-level table, relevant table-column, and a relation between the superior and subordinate columns. Each tuple could (optionally) be further qualified so as to "scope" the relation—for instance, so that the filter should consider only so many levels above the current stack-context, or that the filter only applies if certain other tables also do (or do not) appear in intervening levels—and possibly, even, only in a specific sequence. It would also, of course, be further possible to assign multiple such "sensitivities" to the same target-column. Consider, as an example, a project-management schema, in which both equipment and technicians are assigned to projects; technicians have specific equipment certifications; and schedules apply both to projects and to technicians. In assigning new technicians to a given project, one may wish to automatically "pre-qualify" the dropdown-list of available technicians such that it only includes technicians who are certified on (at least some of) the project's equipment, and who also are currently available during the lifetime of the project

[0220] 11.1.2) "Interactive dropdown controls" are similar, but effect relations between multiple elements within a single mode-display, rather than across context-stack levels. Using the above example, a single many-to-many table might connect technicians to projects; if the table is accessed directly (that is, at the topmost stack-level, rather than by drilling-down to it from the associated project record), then each time the "project"-dropdown is altered, the "technician" dropdown-list would be automatically regenerated according to the above-described criteria. Again, (potentially multiple) specifications per target-column would resemble those for context-sensitive dropdowns, except (of course) that the "superior-level table" and "scoping extensions" would be irrelevant here. Note that although these two dropdown-types are similar—and that, in some cases (namely, where context-sensitive dropdowns utilize only direct drill-down relations), the former could be simulated with the latter—each offers (or lacks) functionality which makes it more suitable for certain types of use

[0221] 11.1.3) "Context-sensitive and interactive column-level security" would allow data-entry fields to "lock" (or unlock) according to values of (and changes in) other data-fields (for instance, once a project has reached a certain "status" designation). Again, specifications could be effected via any of the aforementioned annotational methods, would "attach" to the "target" table-column (i.e., the column whose security is being mediated), and would resemble those for context-sensitive and interactive dropdowns, respectively, except that the "relation" specification would be supplanted by a Boolean evaluation

on the controlling data-field. Note that this same mechanism is easily generalized further to support the toggling of arbitrary column-level constraints (by adding a "constraint definition" field to the specification tuple).

[0222] 11.2) Triggering of custom software subprocesses—on the front- and/or back-end—under specified data conditions and/or at specified system-transition events, such as the "data-change justification" pop-up mechanism described above in detail [0226]

[0223] 12) Various mechanisms for enhancing web-client (or client/server) user-interface performance and functionality can be introduced, comprising:

[0224] 12.1) "Buffered" dropdown controls, which maintain their own separate connections to the back-end RDBMS, and allow the screen display to be rendered before their dropdown lists have been completely populated. Such dropdowns can further be made "typeable", so that a user could begin typing a desired value and "home-in" on matching list-entries; in this case, list-retrieval from the RDBMS can by dynamically revised to retrieve a successively smaller (i.e., closer-matching) result-set.

[0225] 12.2) "Caching" or "sharing" of duplicate dropdown lists, when such lists are lengthy and their retrieval significantly impacts front-end performance and network traffic. For instance, the user-stamping fields described above (Entered_By_Users_Key and Modified_By_Users_Key) generally appear together within the same tables, always share identical dropdown lists, and can (potentially) grow quite long over time; logic to retrieve the shared list once from the RDBMS—rather than twice—for use within both dropdown controls can effect meaningful gains in system responsiveness.

[0226] 12.3) "Back-link" support, to provide functionality similar to that of the standard web-browser "back" button, but without violating the integrity of the user-session or the hierarchical context stack.

[0227] 12.4) "Bookmarking" support, to provide compatibility with standard web-browser "bookmarks" or "favorites" functions: By clicking a special button or link, users can re-render their current display with a re-formed URL, which completely describes the current user-session and context-stack (or, alternatively, a limited and "cauterized" subset of same) so as to allow bookmark-based return to an equivalent display at a later date.

[0228] 13) Although the exemplary embodiment comprises a stand-alone application which interacts (on a client/server basis) with a back-end RDBMS, it may in some circumstances become desirable instead to integrate some or all of the features of the exemplary embodiment directly into said RDBMS product (or a tightly-coupled extension to or utility for same). Of course, any such alternative embodiment would still conform to the principles of the described invention.

[0229] Finally, the implementation described herein could be further varied in numerous respects, but still be within the principles herein illustrated. For instance, while the reference implementation uses a World Wide Web presentation mechanism, a more conventional client-server or native-GUI system could instead be delivered. Also, while the reference implementation depends on adherence to certain structural requirements and naming conventions in the design of any underlying or "target" schema (comprising the use of a single unique, auto-generated primary-key field for every table; the existence of a supporting "sequence" [i.e., reference-implementation RDBMS mechanism for auto-generating primary keys] for every table, and that each sequence be named for its corresponding table plus a "SEQ" suffix; the reservation of "VIEW"-suffixed names across the entire table/view namespace [for use by auto-generated system views]; the use of certain column-name suffixes as alternatives to or substitutes for direct datatype- or other attribute-driven discovery [such as a "FLAG" suffix to connote "yes/no" or "binary" fields, or a "DATE" suffix to indicate time/date data]; and a specific complement of security-related tables, as described below), such requirements and conventions can be easily supplanted, circumvented, or removed, and do not in any way define or limit the scope of the invention.

Run-Time Environment for the Schemalive Reference Implementation

Overview

[0230] The following is specific to the Schemalive Reference Implementation (SRI). The SRI is a web application which conforms to Sun Microsystems' PEE (Java 2 Enterprise Edition) Platform, which in turn incorporates the JSP (Java Server Pages) 1.2, Servlet 2.3, and JDBC (Java Database Connectivity) 2.0 specifications on which the SRI explicitly depends. More information on the structure of web applications can be found at jcp.org/aboutJava/communityprocess/first/jsr053/index.html. The web application can be placed in any J2EE-compliant container (i.e., application-server software), including such products as BEA WebLogic, Macromedia JRun, and Apache Tomcat.

Directory Structure

[0231] A root directory named Schemalive is required; the system's JSP files and static content (i.e., images) are located in this directory. A subdirectory Schemalive/WEB-INF is also required, and must contain a file named web.xml, which is the deployment descriptor (see below) for the application. Supporting classes for the JSP are located in a subdirectory Schemalive/WEB-INF/classes. The web.xml references the application's custom tag libraries (see below) through tag library descriptor files. These XML descriptors are located in a subdirectory Schemalive/WEB-INF/taglib, and have a .tld ile extension. Following is a tree diagram for the SRI directory structure:

```
+Schemalive
    -AddEditForm.jsp
    -BalloonHelp.jsp
    -Browse.jsp
    -DataDictionary.jsp
    -DoAddEdit.jsp
    -DoViewGenerator.jsp
    -Error500.jsp
    -ExpiredSession.jsp
    -OutofSequence.jsp
    -showSession.jsp
    +common
        -EmptyParamCheck.jsp
        -EntryPoints.jsp
```

-continued

```
                -GlobalFooter.jsp
                -GlobalHeaderHTML.jsp
                -GlobalHeaderJavascript.jsp
                -GlobalHeaderVARS.jsp
    +images
        -logo.gif
        -logo-width.gif
    +WEB-INF
        -web.xml
        +classes
            -Connection.properties
            +common
                -Debug.class
            +dbUtils
                -CustomCaps.class
                -CustomDrillDown.class
                -CustomDropDown.class
                -CustomDropDownComponent.class
                -DataDictionary.class
                -DataDictionaryServlet.class
                -DataDictionaryTD.class
                -MasterDetail.class
                -MasterDetailServlet.class
                -SQLUtil.class
                -TableDescriptor.class
                -ViewGenerator.class
            +HTMLUtils
                -Balloon.class
                -BalloonHelp.class
                -TableDescriptorDisplay.class
            +sessionUtils
                -ManageSession.class
                -StackElement.class
                -StackTag.class
                -StackTagExtraInfo.class
            +tagUtils
                -ViewTag.class
                -ViewTagExtraInfo.class
        +taglib
            -stack.tld
            -view.tld
```

Deployment Descriptor

[0232] The deployment descriptor (web.xml) is an XML (eXtensible Markup Language) file which contains all pertinent configuration information for running the web application. The SRI relies on the following portions of the deployment descriptor: servlet definitions; tag library references; and security constraints. The XML parsing rules for this file are contained in a DTD (Document Type Definition) which can be found at java.sun.com/j2ee/dtds/web-app.sub.—2.sub.—2.dtd. Refer to the JSP specification (above) for more information on deployment descriptors.

Servlet Definitions

[0233] The SRI incorporates a number of utility servlets (server-side Java applets which conform to the CGI specification). Servlets are identified in a <servlet> section within web.xml. A name is assigned to each servlet (which is used in creating a servlet mapping, described below), and this name is equated with the appropriate class-file name (specified relative to the Schemalive/WEB-INF/classes subdirectory). For example, a given servlet might be identified as follows:

```
<servlet>
    <servlet-name>DataDictionaryServlet</servlet-name>
    <servlet-class>
        dbUtils.DataDictionaryservlet
    </servlet-name>
</servlet>
```

[0234] By this definition, the following path should exist:

[0235] Schemalive/WEB-INF/classes/dbUtils/DataDictionaryServlet.class

[0236] Note that the <servlet-name> does not represent the actual URL (Uniform Resource Locator) for the servlet; a separate mapping from <servlet-name> to URL occurs in a <servlet-mapping> section:

```
<servlet-mapping>
    <servlet-name>DataDictionaryServlet</servlet-name>
    <url-pattern>DataDictionaryServlet</servlet-name>
</servlet-mapping>
```

[0237] By this definition (and assuming the root directory is Schemalive), the URL:

[0238] <host name>:<port>/Schemalive/DataDictionaryServlet

would cause the J2EE container to execute the code found in

[0239] Schemalive/WEV-INF/classes/dbUtils/DataDictionaryServlet.class

Tag Library References

[0240] A tag library contains Java code that implements custom HTML tags for use within JSPs. When the JSP engine encounters such tags, it makes corresponding Java calls into the tag libraries. For more information, refer to the JSP specification.

[0241] A<taglib>section within web.xml maps a URI (as used from within the JSP) to a tag library descriptor (which contains information about the associated class name, method calls, tag parameters). Below is a sample <taglib>section:

```
<taglib>
    <taglib-uri>view</taglib-uri>
    <taglib-location>WEB-INF/taglib/view.tld</taglib-
location>
</taglib>
```

[0242] See java.sun.com/j2ee/dtds/web-jsptaglib.sub.—1.sub.—1.dtd for the XML DTD for taglib.

[0243] The following is the contents of Schemalive/WEB-INF/taglib/view.tld:

```
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.2</jspversion>
  <tag>
    <name>setVars</name>
    <tagclass>tagUtils.ViewTag</tagclass>
    <teiclass>tagUtils.ViewTagExtraInfo</teiclass>
    <bodycontent>JSP</bodycontent>
    <attribute>
      <name>defaultEntryPoint</name>
```

-continued

```
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
          </attribute>
          <attribute>
            <name>dbName</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
          </attribute>
          <attribute>
            <name>dbConn</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
          </attribute>
        </tag>
      </taglib>
```

[0244] The important parts are the <name>, <tagclass>, and <attribute> tags. The classes referenced in <taglclass> must lie along the J2EE-container's CLASSPATH (note that the Schemalive/WEB-INF/classes directory is automatically included in the CLASSPATH). Combined with <taglib-uri>, there is enough information now to use the custom tag within a JSP. One such invocation would look like this:

```
    <view:setVars defaultEntryPoint="<%= entryPoints[0] %>"
dbName="
        <%= dbName %>" dbConn="<%= dbConnName %>">
    </view:setVars>
```

[0245] Notice the use of <taglib-uri>, <name>, and <attributes> within the custom tag. Also, it is perfectly legal to use JSP inline variables, such as <%=entryPoints[0]%>, as the example shows.

Security Constraints

[0246] web.xml contains information about how the SRI web application should handle security. This includes specifying what to secure, and how—as well as who can access the application (which is governed by the role names to which the user is assigned). The assignment of users to roles, however, is the responsibility of the J2EE container, and is handled differently by the different containers. The <security-constraint>section controls what is protected, and establishes the corresponding role. name, while the <login-config>section establishes the user-authentication method. Here is a sample:

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Schemalive</web-resource-name>
        <url-pattern>/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>Schemalive</role-name>
    </auth-constraint>
</security-constraint>
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Schemalive</realm-name>
</login-config>
```

[0247] Within the <web-resource-collection>section, the <url-pattern> tag protects the entire application (i.e., "/*") for the GET and POST methods. The <auth-constraint> tag references a role named Schemalive; somewhere within the container's configuration, this role is defined and a set of userids and passwords associated with it. The 5<login-config>section establishes BASIC as the authentication method; this is what will cause the userid/password prompt to pop-up when first accessing the site.

Connection Pooling

[0248] The SRI accomplishes database connectivity through the use of connection pooling, as defined in the JDBC 2.0 specification. (For documentation, see java.sun.com/j2se/1.3/docs/guide/jdbc/index.html.)

[0249] In connection pooling, a specified number of connections are pre-made to the underlying RDBMS (Oracle, in the reference implementation) at container start-up time. Connections are "borrowed"—that is, checked in and out of this pool—by program threads on an as-needed basis, without being opened, initialized, closed each time. This provides a dramatic improvement in the application's performance. The mechanics of the connection pool are largely hidden from the software; the standard API calls for opening and closing connections are used, although in actuality the corresponding connections are merely being checked in and out of the pool. The particular interfaces used for connection pooling can be found in the API documentation at java.sun.com/products/jdbc/jdbc20.stdext.javadoc/. (The pertinent classes are javax.sql.ConnectionPoolDataSource and javax.sql.PooledConnection.)

[0250] A static handle to the connection pool is managed through the dbUtils.SQLUtil class, which is implemented in Schemalive/WEB-INF/classes/dbUtils/SQLUtil.java. This class obtains handles to pool connections using the Oracle JDBC 2.0 driver interface; the Javadocs for this API can be found at download.oracle.com.otn/utilities drivers/jdbc/817/java-doc.tar.

[0251] A file named Schemalive/WEB-INF/classes/Connection.properties will need to be customized for each particular installation. JDBCURL contains a (properly formatted) string to reference the Oracle database-server instance. The SRI currently references the Type 2 JDBC driver, and the corresponding URL is in the formal jdbc.oracle:oc18:@<tns name>. The user and pwd properties refer to the credentials the SRI will use for database access; if/when these values need to change, the server must be restarted in order for those changes to take effect.

Run-Time Maintenance

[0252] To enhance system performance (by reducing the need for real-time database qtleries), the SRI maintains two caches of information.

[0253] The first is called the DataDictionary, and contains all of the metadata derived by interrogating the schema (comprising table and column names, column datatypes and sizes, referential-integrity constraints, check constraints, and view definitions). The second is called BalloonHelp, and contains all of the help information specified in the base-tables HELP OBJECT and HELP_SCHEMA.

[0254] When changes are made to the schema structure, or to the records in the help tables, these cached objects must (variously) be refreshed. This can be done dynamically, without having to restart the container.

[0255] The DataDictionary is rebuilt by referencing the JSP DataDictionary.jsp. There are three options when rebuilding the DataDictionary: Only, Views (with check), and Views (without check). The "Only" option simply rebuilds the DataDictionary object (i.e., re-interrogates the database) without rebuilding any (system-generated) views. The other two modes regenerate these views on the fly; the "with check" mode checks to see if a given view (for a corresponding table) already exists, and rebuilds the view only if it is not found. The "without check" option does a brute-force rebuild of all system-generated views, regardless of whether or not they are already defined.

[0256] Note that while the DataDictionary is being rebuilt (which can be a lengthy process, depending on the size of the schema), users will be blocked from accessing the application.

[0257] BalloonHelp is rebuilt by referencing the JSP BalloonHelp.jsp. The current contents of the BalloonHelp object are displayed along with a link to rebuild. When the link is clicked, the cached object is refreshed from the base-tables.

[0258] Changes that are stored to these cached objects are immediately reflected within the application.

[0259] Because of its adherence to various open-standard specifications, the SRI is not dependent on anyone container, but rather, can operate in any J2EE compliant container. The only customization that should be required to run the SRI in a particular environment are the variables (mentioned above and) defined within the Schemalive/WEB-INF/classes/dbUtils/SQLUtil.java file.

[0260] While a number of embodiments have been described in detail, it will be apparent to those skilled in the art that the principles of the invention are realizable by other implementations, structures, and configurations without departing from the scope and spirit of the invention, as defined in the appended claims.

1. (canceled)

2. A method for automatically generating a user interface, operating under control of a computer processor, for working with the data within a relational database, wherein the database is described by a data model comprising a plurality of tables, constraints, and relationships, and is stored within a relational database management system (RDBMS) accessible to the computer processor, the method comprising:

(a) scanning the database to determine the tables, constraints, and relationships of the data model;

(b) creating machine representations of the tables, constraints, and relationships; and

(c) constructing from the representations a corresponding client application that provides:

(i) a connection to the database;

(ii) displays for creating, retrieving, updating, and deleting data within one or more of the tables; and

(iii) mechanisms for representing, managing, and navigating the relationships between data records across related tables, wherein constructing the corresponding client application does not require any incremental human intervention on a per table basis.

3. The method of claim **2**, wherein constructing the corresponding client application does not generate any table-specific code.

4. The method of claim **2**, wherein constructing the corresponding client application does not generate any database-specific code.

5. A computer-implemented system for automatically generating a user interface for working with the data within a relational database, wherein the database is described by a data model comprising a plurality of tables, constraints, and relationships, and is stored within a relational database management system (RDBMS) accessible to a computer processor of a server, the server further comprising:

(a) machine-readable routines for scanning the database to determine the tables, constraints, and relationships of the data model;

(b) machine-readable routines for creating machine representations of the tables, constraints, and relationships; and

(c) machine-readable routines for constructing from the representations a corresponding client application that provides:

(i) a connection to the database;

(ii) displays for creating, retrieving, updating, and deleting data within one or more of the tables; and

(iii) mechanisms for representing, managing, and navigating the relationships between data records across related tables, wherein constructing the corresponding client application does not require any incremental human intervention on a per table basis.

6. The system of claim **5**, wherein constructing the corresponding client application does not generate any table-specific code.

7. The system of claim **5**, wherein constructing the corresponding client application does not generate any database-specific code.

8. A computer-readable storage medium containing a set of instructions for a general purpose computer, for automatically generating a user interface for working with the data within a relational database, wherein the database is described by a data model comprising a plurality of tables, constraints, and relationships, said set of instructions comprising:

(a) a routine for scanning the database to determine the tables, constraints, and relationships of the data model;

(b) a routine for creating machine representations of the tables, constraints, and relationships; and

(c) a routine for constructing from the representations a corresponding client application that provides:

(i) a connection to the database;

(ii) displays for creating, retrieving, updating, and deleting data within one or more of the tables; and

(iii) mechanisms for representing, managing, and navigating the relationships between data records across related tables, wherein constructing the corresponding client application does not require any incremental human intervention on a per table basis.

9. The computer-readable medium of claim **8**, wherein constructing the corresponding client application does not generate any table-specific code.

10. The computer-readable medium of claim **8**, wherein constructing the corresponding client application does not generate any database-specific code.

* * * * *